

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО  
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «САМАРСКИЙ  
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ имени  
академика С.П. КОРОЛЁВА»

КАФЕДРА «ТЕХНИЧЕСКАЯ КИБЕРНЕТИКА»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

«Объектно-ориентированное программирование»

ЛАБОРАТОРНАЯ РАБОТА НОМЕР 4

Студент \_\_\_\_\_ Горейнов Д.В.

Группа \_\_\_\_\_ 6301-030301D

Проверил \_\_\_\_\_ Борисов Д.С

Оценка \_\_\_\_\_

## Задание номер 1

В классах `ArrayTabulatedFunction` и `LinkedListTabulatedFunction` добавил конструкторы, получающие сразу все точки функции в виде массива объектов типа `FunctionPoint`. Если точек задано меньше двух, или если точки в массиве не упорядочены по значению абсциссы, конструкторы должны выбрасывать исключение `IllegalArgumentException`. При написании конструкторов обеспечил корректную инкапсуляцию. (фото 1-2)

```
public LinkedListTabulatedFunction(FunctionPoint[] points) throws IllegalArgumentException {
    if (points.length < 2) {
        throw new IllegalArgumentException("Number of points is less than 2");
    }

    // Проверка упорядоченности точек
    for (int i = 1; i < points.length; i++) {
        if (points[i].getX() <= points[i - 1].getX()) {
            throw new IllegalArgumentException("Points are not ordered by X coordinate");
        }
    }

    head = new FunctionNode();
    head.setNext(head);
    head.setPrev(head);
    pointsCount = 0;

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

```
public ArrayTabulatedFunction(FunctionPoint[] points) {

    if (points.length < 2) {
        throw new IllegalArgumentException("Number of points is less than 2");
    }

    for (int i = 1; i < points.length; i++) {
        if (points[i].getX() <= points[i - 1].getX()) {
            throw new IllegalArgumentException("Points are not ordered by X coordinate");
        }
    }

    this.pointsCount = points.length;
    this.points = new FunctionPoint[pointsCount];
    for (int i = 0; i < pointsCount; i++) {
        this.points[i] = new FunctionPoint(points[i].getX(), points[i].getY());
    }
}
```

Фото 1-2

## Здание номер 2

В пакете `functions` создал интерфейс `Function`, описывающий функции одной переменной и содержащий следующие методы:

- `public double getLeftDomainBorder()` – возвращает значение левой границы области определения функции;
- `public double getRightDomainBorder()` – возвращает значение правой границы области определения функции;

- `public double getFunctionValue(double x)` – возвращает значение функции в заданной точке.

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

```

1  package functions;
2
3  public interface Function {
4      double getLeftDomainBorder();
5      double getRightDomainBorder();
6      double getFunctionValue(double x);
7
8  }
9

```

Фото 3

```

1  package functions;
2
3  public interface TabulatedFunction extends Function{
4
5      void deletePoint(int index) throws FunctionPointIndexOutOfBoundsException;
6      void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;
7      int getPointsCount();
8      FunctionPoint getPoint(int index) throws FunctionPointIndexOutOfBoundsException;
9      void setPoint(int index, FunctionPoint point) throws FunctionPointIndexOutOfBoundsException, InappropriateFunctionPointException;
10     double getPointX(int index) throws FunctionPointIndexOutOfBoundsException;
11     void setPointX(int index, double x) throws FunctionPointIndexOutOfBoundsException, InappropriateFunctionPointException;
12     double getPointY(int index) throws FunctionPointIndexOutOfBoundsException;
13     void setPointY(int index, double y) throws FunctionPointIndexOutOfBoundsException;
14
15 }
16

```

Фото 4

### Задание 3

Создал пакет `functions.basic`, в нём описаны классы ряда функций, заданных аналитически: `cos`, `log`, `sin`, `tan`, `exp` (фото 5)






 Cos.java	19.11.2017 20:24	Исходный файл J...	1 КБ
 Exp.java	10.11.2025 16:20	Исходный файл J...	1 КБ
 Log.java	22.11.2017 22:38	Исходный файл J...	1 КБ
 Sin.java	22.11.2017 22:35	Исходный файл J...	1 КБ
 Tan.java	15.11.2017 23:00	Исходный файл J...	1 КБ

Фото 5

Создал класс `TrigonometricFunction`, реализующий интерфейс `Function` и описывающий методы получения границ области определения. (фото 6).

```
1 package functions;
2
3 public class TrigonometricFunction implements Function{
4     public double getLeftDomainBorder()
5     {
6         return Double.NEGATIVE_INFINITY;
7     }
8     public double getRightDomainBorder()
9     {
10        return Double.POSITIVE_INFINITY;
11    }
12
13    public double getFunctionValue(double x)
14    {
15        return 0;
16    }
17 }
```

Фото 6

Создал наследующие от него публичные классы `Sin`, `Cos` и `Tan`, объекты которых вычисляют, соответственно, значения синуса, косинуса и тангенса. Для получения значений следует воспользоваться методами `Math.sin()`, `Math.cos()` и `Math.tan()` (фото 7-9)

`@Override`— встроенная аннотация в языке Java, которая указывает компилятору, что текущий метод переопределяет метод из суперкласса (родительский класс)

```
package functions.basic;

import functions.TrigonometricFunction;

public class Sin extends TrigonometricFunction{

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

```

1 package functions.basic;
2
3 import functions.TrigonometricFunction;
4
5
6 public class Cos extends TrigonometricFunction {
7
8     @Override
9     public double getFunctionValue(double x)
10    {
11        return Math.cos(x);
12    }
13 }

```

```

1 package functions.basic;
2
3 import functions.TrigonometricFunction;
4
5
6 public class Tan extends TrigonometricFunction {
7
8     @Override
9     public double getFunctionValue(double x)
10    {
11        return Math.tan(x);
12    }
13 }

```

Фото 7-9

## Задание 4

Создал пакет functions.meta, в нём описаны классы функций, позволяющие комбинировать функции: sum, mult, power, scale, shift, composition.

Sum, объекты которого представляют собой функции, являющиеся суммой двух других функций.

Mult, объекты которого представляют собой функции, являющиеся произведением двух других функций.

Power, объекты которого представляют собой функции, являющиеся степенью другой функции

Scale, объекты которого описывают функции, полученные из исходных функций путём масштабирования вдоль осей координат.

Shift, объекты которого описывают функции, полученные из исходных функций путём сдвига вдоль осей координат.

Composition, объекты которого описывают композицию двух исходных функций  
(Фото 10-15)

```
1 package functions.meta;
2
3 import functions.Function;
4
5 public class Sum implements Function
6 {
7     Function a,b;
8     public Sum(Function a, Function b)
9     {
10         this.a=a;
11         this.b=b;
12     }
13
14     @Override
15     public double getLeftDomainBorder()
16     {
17         return Math.max(a.getLeftDomainBorder(), b.getLeftDomainBorder());
18     }
19
20     @Override
21     public double getRightDomainBorder()
22     {
23         return Math.min(a.getRightDomainBorder(), b.getRightDomainBorder());
24     }
25
26     @Override
27     public double getFunctionValue(double x)
28     {
29         return a.getFunctionValue(x)+b.getFunctionValue(x);
30     }
31 }
32 }
```

```
1 package functions.meta;
2
3 import functions.Function;
4
5 public class Mult implements Function
6 {
7     Function a,b;
8
9     public Mult(Function a,Function b)
10
11     {
12         this.a=a;
13         this.b=b;
14     }
15
16
17     @Override
18     public double getLeftDomainBorder()
19     {
20         return Math.max(a.getLeftDomainBorder(),b.getLeftDomainBorder());
21     }
22
23     @Override
24     public double getRightDomainBorder()
25     {
26         return Math.min(a.getLeftDomainBorder(),b.getLeftDomainBorder());
27     }
28
29     @Override
30     public double getFunctionValue(double x)
31     {
32         return a.getFunctionValue(x)*b.getFunctionValue(x);
33     }
34
35
36
37 }
```

```
1  package functions.meta;
2  import functions.Function;
3
4  public class Power implements Function {
5
6      Function a;
7      double b;
8
9      public Power(Function a, double b)
10     {
11         this.a=a;
12         this.b=b;
13     }
14
15     @Override
16     public double getLeftDomainBorder()
17     {
18         return a.getLeftDomainBorder();
19     }
20
21     @Override
22     public double getRightDomainBorder()
23     {
24         return a.getRightDomainBorder();
25     }
26
27     @Override
28     public double getFunctionValue(double x)
29     {
30         return Math.pow(a.getFunctionValue(x),b);
31     }
32 }
```



```
1 package functions.meta;
2 import functions.Function;
3
4 public class Scale implements Function {
5     Function a;
6     double kx,ky;
7
8     public Scale(Function a, double kx,double ky)
9     {
10         this.a=a;
11         this.kx=kx;
12         this.ky=ky;
13     }
14
15     @Override
16     public double getLeftDomainBorder()
17     {
18         return a.getLeftDomainBorder();
19     }
20
21     @Override
22     public double getRightDomainBorder()
23     {
24         return a.getRightDomainBorder();
25     }
26
27     @Override
28
29     public double getFunctionValue(double x)
30     {
31         return a.getFunctionValue(x*kx)*ky;
32     }
33 }
34
35
```

```
1  package functions.meta;
2
3  import functions.Function;
4
5  public class Shift implements Function
6  {
7      Function a;
8      double plus_x, plus_y;
9
10     public Shift(Function a,double plus_x,double plus_y)
11     {
12         this.a=a;
13         this.plus_x=plus_x;
14         this.plus_y=plus_y;
15     }
16
17
18     @Override
19     public double getLeftDomainBorder()
20     {
21         return a.getLeftDomainBorder();
22     }
23
24     @Override
25     public double getRightDomainBorder()
26     {
27         return a.getRightDomainBorder();
28     }
29
30     @Override
31     public double getFunctionValue(double x)
32     {
33         return a.getFunctionValue(x+plus_x)+plus_y;
34     }
35
36 }
```

```

1 package functions.meta;
2
3 import functions.Function;
4
5 public class Composition implements Function
6 {
7     Function a,b;
8
9     public Composition(Function a, Function b)
10    {
11        this.a=a;
12        this.b=b;
13    }
14
15    @Override
16    public double getLeftDomainBorder()
17    {
18        return a.getLeftDomainBorder();
19    }
20
21    @Override
22    public double getRightDomainBorder()
23    {
24        return a.getRightDomainBorder();
25    }
26
27    @Override
28    public double getFunctionValue(double x)
29    {
30        return a.getFunctionValue(b.getFunctionValue(x));
31    }
32
33 }

```

Фото 10-15

## Задание 5

В пакете functions создал класс Functions, содержащий вспомогательные статические методы для работы с функциями. Сделал так, чтобы в программе вне этого класса нельзя было создать его объект. Класс содержит следующие методы:

- public static Function shift(Function f, double shiftX, double shiftY) – возвращает объект функции, полученной из исходной сдвигом вдоль осей;
- public static Function scale(Function f, double scaleX, double scaleY) – возвращает объект функции, полученной из исходной масштабированием вдоль осей;
- public static Function power(Function f, double power) – возвращает объект функции, являющейся заданной степенью исходной;
- public static Function sum(Function f1, Function f2) – возвращает объект функции, являющейся суммой двух исходных;

- `public static Function mult(Function f1, Function f2)` – возвращает объект функции, являющейся произведением двух исходных;
- `public static Function composition(Function f1, Function f2)` – возвращает объект функции, являющейся композицией двух исходных.

```
package functions;

import functions.meta.*;

public final class Functions {
    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)
    {
        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);
    }
}
```

Фото 16

## Задание 6

В пакете `functions` создал класс `TabulatedFunctions`, содержащий вспомогательные статические методы для работы с табулированными функциями. Сделал так, чтобы в программе вне этого класса нельзя было создать его объект.

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.

Если указанные границы для табулирования выходят за область определения функции, метод выбрасывает исключение `IllegalArgumentException`.

```
package functions;

import java.io.*;

public final class TabulatedFunctions {

    public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount) {
        // Проверяем, что интервал внутри области определения
        if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {
            throw new IllegalArgumentException("Заданный интервал выходит за границы области определения функции");
        }

        if (pointsCount < 2) {
            throw new IllegalArgumentException("Количество точек должно быть не менее 2");
        }

        FunctionPoint[] points = new FunctionPoint[pointsCount];
        double step = (rightX - leftX) / (pointsCount - 1);

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

        return new ArrayTabulatedFunction(points);
    }
}
```

Фото 17

## Задание 7

В класс `TabulatedFunctions` добавьте следующие методы.

Метод вывода табулированной функции в байтовый поток `public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out)` в указанный поток выводит значения, по которым потом можно будет восстановить табулированную функцию, а именно количество точек в ней и значения координат точек.

Метод ввода табулированной функции из байтового потока `public static TabulatedFunction inputTabulatedFunction(InputStream in)` считывает из указанного потока данные о табулированной функции, создаёт и настраивает её объект и возвращает его из метода.

Метод записи табулированной функции в символьный поток `public static void writeTabulatedFunction(TabulatedFunction function, Writer out)` в указанный поток выводит значения, по которым потом можно будет восстановить табулированную функцию, а именно количество точек в ней и значения координат точек.

Метод чтения табулированной функции из символьного потока `public static TabulatedFunction readTabulatedFunction(Reader in)` считывает из указанного потока данные о табулированной функции, создает и настраивает её объект и возвращает его из метода.

Подумайте и обоснуйте, как следует в этих методах поступить с возникающим исключением `IOException`:

Объявляется `throws IOException`

Пробрасываются исключение вызывающему коду

Поток не следует закрывать, так как если бы метод закрыл поток, клиент получил бы исключение при попытке дописать данные.

```
public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out) throws IOException {
    DataOutputStream dataOut = new DataOutputStream(out);
    int pointCount = function.getPointsCount();
    dataOut.writeInt(pointCount);

    for (int i = 0; i < pointCount; i++) {
        dataOut.writeDouble(function.getPointX(i));
        dataOut.writeDouble(function.getPointY(i));
    }
    dataOut.flush();
}

public static TabulatedFunction inputTabulatedFunction(InputStream in) throws IOException {
    DataInputStream dataIn = new DataInputStream(in);
    int pointCount = dataIn.readInt();
    FunctionPoint[] data = new FunctionPoint[pointCount];

    for (int i = 0; i < pointCount; i++) {
        double x = dataIn.readDouble();
        double y = dataIn.readDouble();
        data[i] = new FunctionPoint(x, y);
    }

    return new ArrayTabulatedFunction(data);
}

public static void writeTabulatedFunction(TabulatedFunction function, Writer out) throws IOException {
    PrintWriter writer = new PrintWriter(out);
    int pointCount = function.getPointsCount();
    writer.println(pointCount);

    for (int i = 0; i < pointCount; i++) {
        writer.println(function.getPointX(i));
        writer.println(function.getPointY(i));
    }
    writer.flush();
}

public static TabulatedFunction readTabulatedFunction(Reader in) throws IOException {
    StreamTokenizer st = new StreamTokenizer(in);
    st.nextToken();
    int itemsCount = (int) st.nval;
    FunctionPoint[] data = new FunctionPoint[itemsCount];

    for (int i = 0; i < itemsCount; i++) {
        st.nextToken();
        double x = st.nval;
        st.nextToken();
        double y = st.nval;
        data[i] = new FunctionPoint(x, y);
    }

    return new ArrayTabulatedFunction(data);
}
```

Фото 18

## Задание 8

Синус и косинус:

Sin = 0,000000	TabSin = 0,000000	Cos = 1,000000	TabCos = 1,000000
Sin = 0,099833	TabSin = 0,097982	Cos = 0,995004	TabCos = 0,982723
Sin = 0,198669	TabSin = 0,195963	Cos = 0,980067	TabCos = 0,965446
Sin = 0,295520	TabSin = 0,293945	Cos = 0,955336	TabCos = 0,948170
Sin = 0,389418	TabSin = 0,385907	Cos = 0,921061	TabCos = 0,914355
Sin = 0,479426	TabSin = 0,472070	Cos = 0,877583	TabCos = 0,864608
Sin = 0,564642	TabSin = 0,558234	Cos = 0,825336	TabCos = 0,814862
Sin = 0,644218	TabSin = 0,643982	Cos = 0,764842	TabCos = 0,764620
Sin = 0,717356	TabSin = 0,707935	Cos = 0,696707	TabCos = 0,688404
Sin = 0,783327	TabSin = 0,771888	Cos = 0,621610	TabCos = 0,612188
Sin = 0,841471	TabSin = 0,835841	Cos = 0,540302	TabCos = 0,535972
Sin = 0,891207	TabSin = 0,883993	Cos = 0,453596	TabCos = 0,450633
Sin = 0,932039	TabSin = 0,918022	Cos = 0,362358	TabCos = 0,357141
Sin = 0,963558	TabSin = 0,952051	Cos = 0,267499	TabCos = 0,263648
Sin = 0,985450	TabSin = 0,984808	Cos = 0,169967	TabCos = 0,169931
Sin = 0,997495	TabSin = 0,984808	Cos = 0,070737	TabCos = 0,070437
Sin = 0,999574	TabSin = 0,984808	Cos = -0,029200	TabCos = -0,029056
Sin = 0,991665	TabSin = 0,984808	Cos = -0,128844	TabCos = -0,128549
Sin = 0,973848	TabSin = 0,966204	Cos = -0,227202	TabCos = -0,224761
Sin = 0,946300	TabSin = 0,932175	Cos = -0,323290	TabCos = -0,318254
Sin = 0,909297	TabSin = 0,898147	Cos = -0,416147	TabCos = -0,411747
Sin = 0,863209	TabSin = 0,862441	Cos = -0,504846	TabCos = -0,504272
Sin = 0,808496	TabSin = 0,798488	Cos = -0,588501	TabCos = -0,580488
Sin = 0,745705	TabSin = 0,734535	Cos = -0,666276	TabCos = -0,656704
Sin = 0,675463	TabSin = 0,670582	Cos = -0,737394	TabCos = -0,732920
Sin = 0,598472	TabSin = 0,594072	Cos = -0,801144	TabCos = -0,794171
Sin = 0,515501	TabSin = 0,507908	Cos = -0,856889	TabCos = -0,843917
Sin = 0,427380	TabSin = 0,421745	Cos = -0,904072	TabCos = -0,893664
Sin = 0,334988	TabSin = 0,334698	Cos = -0,942222	TabCos = -0,940984
Sin = 0,239249	TabSin = 0,236716	Cos = -0,970958	TabCos = -0,958261
Sin = 0,141120	TabSin = 0,138735	Cos = -0,989992	TabCos = -0,975537
Sin = 0,041581	TabSin = 0,040753	Cos = -0,999135	TabCos = -0,992814

Сумма квадратов синуса и косинуса:

Arg = 0,000000	Value = 1,000000
Arg = 0,100000	Value = 0,975345
Arg = 0,200000	Value = 0,970488
Arg = 0,300000	Value = 0,985429
Arg = 0,400000	Value = 0,984968
Arg = 0,500000	Value = 0,970398
Arg = 0,600000	Value = 0,975624
Arg = 0,700000	Value = 0,999358
Arg = 0,800000	Value = 0,975073
Arg = 0,900000	Value = 0,970586
Arg = 1,000000	Value = 0,985897
Arg = 1,100000	Value = 0,984515
Arg = 1,200000	Value = 0,970314
Arg = 1,300000	Value = 0,975910
Arg = 1,400000	Value = 0,998723

```

Arg = 1,600000 Value = 0,970691
Arg = 1,700000 Value = 0,986371
Arg = 1,800000 Value = 0,984068
Arg = 1,900000 Value = 0,970237
Arg = 2,000000 Value = 0,976203
Arg = 2,100000 Value = 0,998094
Arg = 2,200000 Value = 0,974549
Arg = 2,300000 Value = 0,970802
Arg = 2,400000 Value = 0,986852
Arg = 2,500000 Value = 0,983628
Arg = 2,600000 Value = 0,970167
Arg = 2,700000 Value = 0,976503
Arg = 2,800000 Value = 0,997473
Arg = 2,900000 Value = 0,974298
Arg = 3,000000 Value = 0,970920
Arg = 3,100000 Value = 0,987341

```

Экспонента(символьный поток):

```

Exp = 1,000000      ReadTabExp = 1,000000
Exp = 2,718282      ReadTabExp = 2,718282
Exp = 7,389056      ReadTabExp = 7,389056
Exp = 20,085537     ReadTabExp = 20,085537
Exp = 54,598150     ReadTabExp = 54,598150
Exp = 148,413159    ReadTabExp = 148,413159
Exp = 403,428793    ReadTabExp = 403,428793
Exp = 1096,633158   ReadTabExp = 1096,633158
Exp = 2980,957987   ReadTabExp = 2980,957987
Exp = 8103,083928   ReadTabExp = 8103,083928
Exp = 22026,465795  ReadTabExp = 22026,465795

```

Логарифм(байтовый поток):

```

x=1,0: Ln = 0,000000      ReadTabLn = 0,000000
x=1,9: Ln = 0,641854      ReadTabLn = 0,641854
x=2,8: Ln = 1,029619      ReadTabLn = 1,029619
x=3,7: Ln = 1,308333      ReadTabLn = 1,308333
x=4,6: Ln = 1,526056      ReadTabLn = 1,526056
x=5,5: Ln = 1,704748      ReadTabLn = 1,704748
x=6,4: Ln = 1,856298      ReadTabLn = 1,856298
x=7,3: Ln = 1,987874      ReadTabLn = 1,987874
x=8,2: Ln = 2,104134      ReadTabLn = 2,104134
x=9,1: Ln = 2,208274      ReadTabLn = 2,208274
x=10,0: Ln = 2,302585     ReadTabLn = 2,302585

```

Логарифм от экспоненты:

```

LogExp = 1,000000      SerLogExp = 1,000000
LogExp = 2,000000      SerLogExp = 2,000000
LogExp = 3,000000      SerLogExp = 3,000000
LogExp = 4,000000      SerLogExp = 4,000000
LogExp = 5,000000      SerLogExp = 5,000000
LogExp = 6,000000      SerLogExp = 6,000000
LogExp = 7,000000      SerLogExp = 7,000000
LogExp = 8,000000      SerLogExp = 8,000000
LogExp = 9,000000      SerLogExp = 9,000000
LogExp = 10,000000     SerLogExp = 10,000000

```

Фото 19-21

Вывод:



Плюсы:

Байтовый поток точно сохраняет значения функции и занимает меньше памяти.

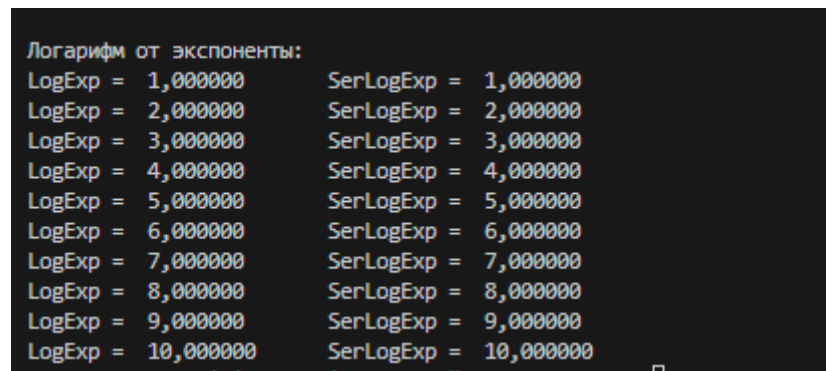
Минусы:

Трудно читать пользователю

```
aced 0005 7372 0020 6675 6e63 7469 6f6e
732e 4172 7261 7954 6162 756c 6174 6564
4675 6e63 7469 6f6e c96a 0fc6 cfc9 8523
0200 0249 000b 706f 696e 7473 436f 756e
745b 0006 706f 696e 7473 7400 1a5b 4c66
756e 6374 696f 6e73 2f46 756e 6374 696f
6e50 6f69 6e74 3b78 7000 0000 0b75 7200
1a5b 4c66 756e 6374 696f 6e73 2e46 756e
6374 696f 6e50 6f69 6e74 3b62 3fa4 1d5f
41dd a702 0000 7870 0000 000b 7372 0017
6675 6e63 7469 6f6e 732e 4675 6e63 7469
6f6e 506f 696e 74b2 f17c b904 cf4b 0f02
0002 4400 0178 4400 0179 7870 3ff0 0000
0000 0000 3ff0 0000 0000 0000 7371 007e
0005 3ffe 6666 6666 6666 3ffe 6666 6666
6666 7371 007e 0005 4006 6666 6666 6666
4006 6666 6666 6666 7371 007e 0005 400d
9999 9999 999a 400d 9999 9999 999a 7371
007e 0005 4012 6666 6666 6666 4012 6666
6666 6666 7371 007e 0005 4016 0000 0000
0000 4016 0000 0000 0000 7371 007e 0005
4019 9999 9999 999a 4019 9999 9999 999a
7371 007e 0005 401d 3333 3333 3333 401d
3333 3333 3333 7371 007e 0005 4020 6666
6666 6666 4020 6666 6666 6666 7371 007e
0005 4022 3333 3333 3333 4022 3333 3333
3333 7371 007e 0005 4024 0000 0000 0000
4024 0000 0000 0000
```

Фото 22

## Задание 9



Логарифм от экспоненты:		
LogExp =	1,000000	SerLogExp = 1,000000
LogExp =	2,000000	SerLogExp = 2,000000
LogExp =	3,000000	SerLogExp = 3,000000
LogExp =	4,000000	SerLogExp = 4,000000
LogExp =	5,000000	SerLogExp = 5,000000
LogExp =	6,000000	SerLogExp = 6,000000
LogExp =	7,000000	SerLogExp = 7,000000
LogExp =	8,000000	SerLogExp = 8,000000
LogExp =	9,000000	SerLogExp = 9,000000
LogExp =	10,000000	SerLogExp = 10,000000

Фото 23

Плюсы Serializable:

- Простота использования
- Автоматическое управление

Минусы: Serializable

- Медленнее

- Низкая безопасность

Плюсы Externalizable:

- Полный контроль
- Быстрый
- Высокая защита

Минусы Externalizable:

- Сложнее реализовать
- Требует ручного управления в методах