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 №4

Студент Луковникова Е.Д

Группа 6301-030301D

Руководитель Борисов Д. С.

Оценка _____

САМАРА 2025

Задание 1

В классах ArrayTabulatedFunction и LinkedListTabulatedFunction добавила 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 выбрасывают исключение IllegalArgumentException.

```
public ArrayTabulatedFunction(FunctionPoint[] points) {
if (points == null || points.length < 2) {
    throw new IllegalArgumentException(s: "Должно быть не менее 2 точек");
}

for (int i = 1; i < points.length; i++) {
    if (points[i].getX() <= points[i-1].getX()) {
        throw new IllegalArgumentException(s: "Точки не упорядочены по X");
    }
}
```

Скрин 1

```
public LinkedListTabulatedFunction(FunctionPoint[] points) {
if (points == null || points.length < 2) {
    throw new IllegalArgumentException(s: "Должно быть не менее 2 точек");
}

for (int i = 1; i < points.length; i++) {
    if (points[i].getX() <= points[i-1].getX()) {
        throw new IllegalArgumentException(s: "Точки не упорядочены по X");
    }
}
```

Скрин 2

Задание 2

В пакете functions создала 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.

Исключила соответствующие методы из интерфейса TabulatedFunction и сделала так, что он расширяет интерфейс Function.

```
package functions;

public interface Function {
    double getLeftDomainBorder();
    double getRightDomainBorder();
    double getFunctionValue(double x);
}
```

Скрин 3

```
package functions;

import java.io.*;

public interface TabulatedFunction extends Function, Serializable{
    int getPointsCount();
    FunctionPoint getPoint(int index);
    void setPoint(int index,FunctionPoint point) throws InappropriateFunctionPointException ;
    double getPointX(int index);
    void setPointX(int index,double x) throws InappropriateFunctionPointException;
    double getPointY(int index);
    void setPointY(int index,double y);
    void deletePoint(int index) throws InappropriateFunctionPointException;
    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;
}
```

Скрин 4

Задание 3

Создала пакет functions.basic, в нём описаны классы ряда функций, заданных аналитически.

Создала в пакете публичный класс Exp, объекты которого вычисляют значение экспоненты. Класс реализует интерфейс Function. Для вычисления экспоненты пользуемся Math.exp(), а для возвращения значений границ области определения – константами из класса Double.

Аналогично, создайте класс Log, объекты которого вычисляют значение логарифма по заданному основанию. Основание передается как параметр конструктора. Для вычисления логарифма пользуемся методом Math.log().

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

Создала наследующие от него публичные классы Sin, Cos и Tan, объекты которых вычисляют значения синуса, косинуса и тангенса. Для получения значений пользуемся методами Math.sin(), Math.cos() и Math.tan().

```java
package functions.basic;

import functions.Function;

public class Exp implements Function {

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

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

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

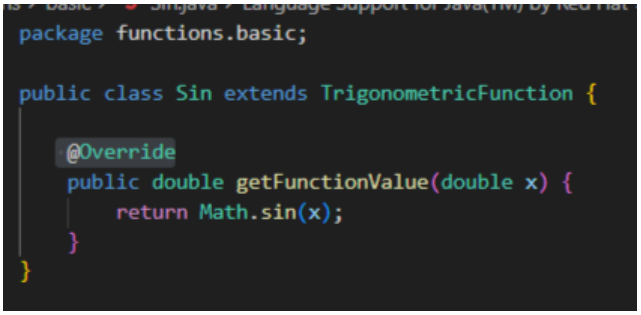
Скрин 5

```java
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;
    }
}
```

Скрин 6

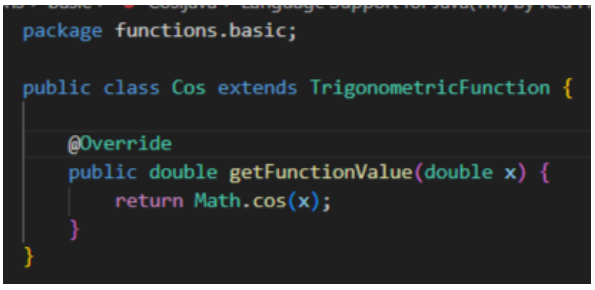
```java
package functions.basic;

import functions.Function;

public class Log implements Function {
    private double base;

    public Log(double base) {
        if (base < 0) {
            throw new IllegalArgumentException(s: "Основание должно быть > 0");
        }
        this.base = base;
    }

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

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

    @Override
    public double getFunctionValue(double x) {
        if (x <= 0) {
            return Double.NaN;
        }
        else{
            return Math.log(x) / Math.log(base);
        }
    }
}
```

Скрин 7



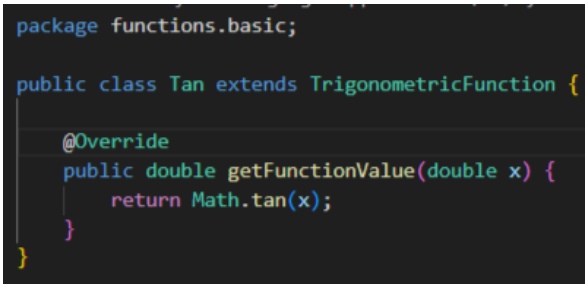
```java
package functions.basic;

public class Sin extends TrigonometricFunction {

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

Скрин 8



```java
package functions.basic;

public class Cos extends TrigonometricFunction {

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

Скрин 9



```java
package functions.basic;

public class Tan extends TrigonometricFunction {

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

Скрин 10

Задание 4

Создала пакет functions.meta, в котором описаны классы функций, позволяющие комбинировать функции.

Создала класс Sum, объекты которого представляют собой функции, являющиеся суммой двух других функций. Класс реализует интерфейс Function. Конструктор класса получает ссылки типа Function на объекты суммируемых функций, а область определения функции получается, как пересечение областей определения исходных функций.

```java
package functions.meta;

import functions.Function;

public class Sum implements Function {
    private Function f1;
    private Function f2;

    public Sum(Function f1, Function f2) {
        if (f1 == null || f2 == null){
            throw new IllegalArgumentException(s: "Функции не могут быть нулевыми");
        }
        this.f1 = f1;
        this.f2 = f2;
    }

    @Override
    public double getLeftDomainBorder() {
        return Math.max(f1.getLeftDomainBorder(), f2.getLeftDomainBorder());
    }

    @Override
    public double getRightDomainBorder() {
        return Math.min(f1.getRightDomainBorder(), f2.getRightDomainBorder());
    }

    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()){
            return Double.NaN;
        }
        else{
            return f1.getFunctionValue(x) + f2.getFunctionValue(x);
        }
    }
}
```

Скрин 11

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

```java
package functions.meta;

import functions.Function;

public class Mult implements Function {
    private Function f1;
    private Function f2;

    public Mult(Function f1, Function f2) {
        if (f1 == null || f2 == null){
            throw new IllegalArgumentException(s: "Функции не могут быть нулевыми");
        }
        this.f1 = f1;
        this.f2 = f2;
    }

    @Override
    public double getLeftDomainBorder() {
        return Math.max(f1.getLeftDomainBorder(), f2.getLeftDomainBorder());
    }

    @Override
    public double getRightDomainBorder() {
        return Math.min(f1.getRightDomainBorder(), f2.getRightDomainBorder());
    }

    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()){
            return Double.NaN;
        }
        else{
            return f2.getFunctionValue(f1.getFunctionValue(x));
        }
    }
}
```

Скрин 12

Создала класс Power, объекты которого представляют собой функции, являющиеся степенью другой функции. Конструктор класса получает ссылку на объекты базовой функции и степень, в которую возводятся её значения. Область определения функции считается совпадающей с областью определения исходной функции.

```
package functions.meta;

import functions.Function;

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

    public Power(Function f, double power) {
        if (f == null){
            throw new IllegalArgumentException(s: "Функция не могут быть нулём");
        }
        this.f = f;
        this.power = power;
    }

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

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

    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()){
            return Double.NaN;
        }
        else{
            return Math.pow(f.getFunctionValue(x), power);
        }

    }
}
```

Скрин 13

Создала класс Scale, объекты которого описывают функции, полученные из исходных функций путём масштабирования вдоль осей координат. Конструктор класса получает ссылку на объект исходной функции, а также коэффициенты масштабирования вдоль оси абсцисс и оси ординат. Область определения функции получается из области определения исходной функции масштабированием вдоль оси абсцисс, а значение функции – масштабированием значения исходной функции вдоль оси ординат.

```
package functions.meta;

import functions.Function;

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

    public Scale(Function f, double scaleX, double scaleY) {
        if (f == null){
            throw new IllegalArgumentException(s: "Функция не могут быть нулём");
        }
        this.f = f;
        this.scaleX = scaleX;
        this.scaleY = scaleY;
    }

    @Override
    public double getLeftDomainBorder() {
        if (scaleX > 0){
            return f.getLeftDomainBorder() * scaleX;
        } else if (scaleX < 0){
            return f.getRightDomainBorder() * scaleX;
        } else {
            return Double.NaN;
        }

    }
```

Скрин 14

```
    @Override
    public double getRightDomainBorder() {
        if (scaleX > 0){
            return f.getRightDomainBorder() * scaleX;
        } else if (scaleX < 0){
            return f.getLeftDomainBorder() * scaleX;
        } else {
            return Double.NaN;
        }
    }

    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()){
            return Double.NaN;
        }
        else{
            return f.getFunctionValue(x * scaleX) * scaleY;
        }

    }
}
```

Скрин 15

Аналогично, создала 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;
    private double shiftY;

    public Shift(Function f, double shiftX, double shiftY) {
        if (f == null){
            throw new IllegalArgumentException(s: "Функция не могут быть нулём");
        }
        this.f = f;
        this.shiftX = shiftX;
        this.shiftY = shiftY;
    }

    @Override
    public double getLeftDomainBorder() {
        return f.getLeftDomainBorder() - shiftX;
    }

    @Override
    public double getRightDomainBorder() {
        return f.getRightDomainBorder() - shiftX;
    }

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

Скрин 16

Также создала класс Composition, объекты которого описывают композицию двух исходных функций. Конструктор класса получает ссылки на объекты первой и второй функции. Область определения функции считается совпадающей с областью определения исходной функции.

```
package functions.meta;

import functions.Function;

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

    public Composition(Function f1, Function f2) {
        if (f1 == null || f2 == null){
            throw new IllegalArgumentException(s: "Функции не могут быть нулевыми");
        }
        this.f1 = f1;
        this.f2 = f2;
    }
```

Скрин 17

```
    @Override
    public double getLeftDomainBorder() {
        return f1.getLeftDomainBorder();
    }

    @Override
    public double getRightDomainBorder() {
        return f1.getRightDomainBorder();
    }

    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()){
            return Double.NaN;
        }
        else{
            return f2.getFunctionValue(f1.getFunctionValue(x));
        }

    }
}
```

Скрин 18

Задание 5

В пакете functions создала класс Functions, содержащий вспомогательные статические методы для работы с функциями. В программе вне этого класса нельзя создать его объект. Класс должен 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.

```java
package functions;

import functions.meta.*;

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

Скрин 19

Задание 6

В пакете functions создала класс TabulatedFunctions, содержащий вспомогательные статические методы для работы с табулированными функциями. В программе вне этого класса нельзя создать его объект.

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 должен выбрасывает исключение IllegalArgumentException.

```
public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount) {
    if (pointsCount < 2) {
        throw new IllegalArgumentException(s: "Требуется не менее 2 точек");
    }
    if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {
        throw new IllegalArgumentException(s: "Заданные границы выходят за область определения");
    }
    if(leftX >= rightX){
        throw new IllegalArgumentException(s: "Левая граница больше или равна правой");
    }
    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));
    }

    return new ArrayTabulatedFunction(points);
}
```

Скрин 20

Задание 7

В класс TabulatedFunctions добавила следующие методы.

Метод вывода табулированной функции в байтовый поток public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out) в указанный поток выводит значения, по которым можно 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, создает 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 можно восстановить табулированную функцию, а именно количество точек в ней и значения координат точек. Значения записываются в строку и разделяются пробелами.

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.

```java
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) {
        FunctionPoint point = function.getPoint(i);
        dos.writeDouble(point.getX());
        dos.writeDouble(point.getY());
    }
    dos.flush();
}

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

    for (int i = 0; i < pointCount; i++) {
        points[i] = new FunctionPoint(dis.readDouble(), dis.readDouble());
    }
    return new ArrayTabulatedFunction(points);
}

public static void writeTabulatedFunction(TabulatedFunction function, Writer out) throws IOException {
    BufferedWriter Writer = new BufferedWriter(out);
    int pointsCount = function.getPointsCount();
    Writer.write(" " + pointsCount);

    for (int i = 0; i < pointsCount; i++) {
        FunctionPoint point = function.getPoint(i);
        Writer.write("\n " + point.getX());
        Writer.write(" " + point.getY());
    }
    Writer.flush();
}
```

Скрин 21

```java
public static TabulatedFunction readTabulatedFunction(Reader in) throws IOException {
    StreamTokenizer st = new StreamTokenizer(in);
    st.nextToken();
    int pointsCount = (int) st.nval;
    FunctionPoint[] points = new FunctionPoint[pointsCount];

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

    return new ArrayTabulatedFunction(points);
}
```

Скрин 22

Задание 8

Создала по одному объекту классов Sin и Cos, выводятся в консоль значения этих функций на отрезке от 0 до π с шагом 0,1.

```
Function fun1 = new Cos();
Function fun2 = new Sin();
double pi = Math.PI;
System.out.println(x: "\nЗначения sin и cos\n");
for (double i = 0; i <= pi; i += 0.1){
    System.out.printf(format: "sin(%.2f) = %.6f \t cos(%.2f) = %.6f\n", i , fun2.getFunctionValue(i), i , fun1.getFunctionValue(i));
}
```

С помощью метода TabulatedFunctions.tabulate() создала табулированные аналоги этих функций на отрезке от 0 до π с 10 точками. Вывела в консоль значения этих функций на отрезке от 0 до π с шагом 0,1 и сравнила со значениями исходных функций.

```
TabulatedFunction TabulatedCos = TabulatedFunctions.tabulate(fun1, leftX: 0, pi, pointsCount: 30);
TabulatedFunction TabulatedSin = TabulatedFunctions.tabulate(fun2, leftX: 0, pi, pointsCount: 30);
System.out.println(x: "\nЗначения табулированных sin и cos\n");
for (double i = 0; i <= pi; i += 0.1){
    System.out.printf(format: "sin(%.2f) = %.6f \t cos(%.2f) = %.6f\n", i ,  TabulatedSin.getFunctionValue(i), i , TabulatedCos.getFunctionValue(i));
}
```

```
System.out.println(x: "\nРассчет модуля разности между табулированными и исходными значениями sin и cos\n");
for (double i = 0; i <= pi; i += 0.1){
    double absSin = Math.abs(TabulatedSin.getFunctionValue(i) - fun2.getFunctionValue(i));
    double absCos = Math.abs(TabulatedCos.getFunctionValue(i) - fun1.getFunctionValue(i));
    System.out.printf(format: "Разность sin = %.6f \t Разность cos = %.6f\n", absSin, absCos);
}
```

С помощью методов класса Functions создала объект функции, являющейся суммой квадратов табулированных аналогов синуса и косинуса. Вывела в консоль значения этой функций на отрезке от 0 до π с шагом 0,1.

```
Function SumOfSquaresSinAndCos = Functions.sum(Functions.power(TabulatedSin, power: 2), Functions.power(TabulatedCos, power: 2));
System.out.println(x: "\nСумма квадратов синуса и косинуса\n");
for (double i = 0; i <= pi; i += 0.1) {
    System.out.printf(format: "sin(%.2f)^2 + cos(%.2f)^2 = %.6f\n", i, i, SumOfSquaresSinAndCos.getFunctionValue(i));
}
```

С помощью метода TabulatedFunctions.tabulate() создала табулированный аналог экспоненты на отрезке от 0 до 10 с 11 точками. С помощью метода TabulatedFunctions.writeTabulatedFunction() вывела его в файл. Далее с помощью метода TabulatedFunctions.readTabulatedFunction() считала табулированную функцию из этого файла. Вывела и сравнила значения исходной и считанной функции на отрезке от 0 до 10 с шагом 1.

```
TabulatedFunction Exp = TabulatedFunctions.tabulate(new Exp(), leftX: 0, rightX: 10, pointsCount: 11);
File f = new File(pathname: "exp.txt");
FileWriter fw = new FileWriter(f);
TabulatedFunctions.writeTabulatedFunction(Exp, fw);
fw.close();
FileReader fr = new FileReader(f);
TabulatedFunction TabExp = TabulatedFunctions.readTabulatedFunction(fr);

System.out.println(x: "\nЗначение экспонент\n");
for (int i = 0; i < 11; i++){
    System.out.printf(format: "Exp = %.6f \t TabExp = %.6f\n", Exp.getFunctionValue(i), TabExp.getFunctionValue(i));
}
```

С помощью метода TabulatedFunctions.tabulate() создала табулированный аналог логарифма по натуральному основанию на отрезке от 0 до 10 с 11 точками. С помощью метода TabulatedFunctions.outputTabulatedFunction() вывела его в файл. Далее с помощью метода TabulatedFunctions.inputTabulatedFunction() считала табулированную функцию из этого

файла. Вывела и сравнила значения исходной и считанной функции на отрезке от 0 до 10 с шагом 1.

```
TabulatedFunction ln = TabulatedFunctions.tabulate(new Log(Math.E), leftX: 1, rightX: 10, pointsCount: 11);
File f2 = new File(pathname: "ln.txt");
FileOutputStream fw2 = new FileOutputStream(f2);
TabulatedFunctions.outputTabulatedFunction(ln, fw2);
fw2.close();
FileInputStream fr2 = new FileInputStream(f2);
TabulatedFunction TabLn = TabulatedFunctions.inputTabulatedFunction(fr2);

System.out.println(x: "\nЗначение натурального логарифма\n");
for (int i = 1; i < 11; i++){
    System.out.printf("Ln(" + i + ") = %.6f \t TabLn(" + i + ") = %.6f\n", ln.getFunctionValue(i), TabLn.getFunctionValue(i));
}
```

Задание 9

Сделала так, чтобы объекты всех классов, реализующих интерфейс TabulatedFunction, были сериализуемыми.

Проверила работу написанных классов. С помощью метода TabulatedFunctions.tabulate() и метода класса Functions создала табулированный аналог логарифма по натуральному основанию, взятого от экспоненты на отрезке от 0 до 10 с 11 точками. Сериализовала полученный объект в файл. Далее десериализовала табулированную функцию из этого файла. Вывела значения исходной и считанной функции на отрезке от 0 до 10 с шагом 1.

Результат работы

```
Значения sin и cos

sin(0,00) = 0,000000     cos(0,00) = 1,000000
sin(0,10) = 0,099833     cos(0,10) = 0,995004
sin(0,20) = 0,198669     cos(0,20) = 0,980067
sin(0,30) = 0,295520     cos(0,30) = 0,955336
sin(0,40) = 0,389418     cos(0,40) = 0,921061
sin(0,50) = 0,479426     cos(0,50) = 0,877583
sin(0,60) = 0,564642     cos(0,60) = 0,825336
sin(0,70) = 0,644218     cos(0,70) = 0,764842
sin(0,80) = 0,717356     cos(0,80) = 0,696707
sin(0,90) = 0,783327     cos(0,90) = 0,621610
sin(1,00) = 0,841471     cos(1,00) = 0,540302
sin(1,10) = 0,891207     cos(1,10) = 0,453596
sin(1,20) = 0,932039     cos(1,20) = 0,362358
sin(1,30) = 0,963558     cos(1,30) = 0,267499
sin(1,40) = 0,985450     cos(1,40) = 0,169967
sin(1,50) = 0,997495     cos(1,50) = 0,070737
sin(1,60) = 0,999574     cos(1,60) = -0,029200
sin(1,70) = 0,991665     cos(1,70) = -0,128844
sin(1,80) = 0,973848     cos(1,80) = -0,227202
sin(1,90) = 0,946300     cos(1,90) = -0,323290
sin(2,00) = 0,909297     cos(2,00) = -0,416147
sin(2,10) = 0,863209     cos(2,10) = -0,504846
sin(2,20) = 0,808496     cos(2,20) = -0,588501
sin(2,30) = 0,745705     cos(2,30) = -0,666276
sin(2,40) = 0,675463     cos(2,40) = -0,737394
sin(2,50) = 0,598472     cos(2,50) = -0,801144
sin(2,60) = 0,515501     cos(2,60) = -0,856889
sin(2,70) = 0,427380     cos(2,70) = -0,904072
sin(2,80) = 0,334988     cos(2,80) = -0,942222
sin(2,90) = 0,239249     cos(2,90) = -0,970958
sin(3,00) = 0,141120     cos(3,00) = -0,989992
sin(3,10) = 0,041581     cos(3,10) = -0,999135
```

```
Значения табулированных sin и cos

sin(0,00) = 0,000000     cos(0,00) = 1,000000
sin(0,10) = 0,099805     cos(0,10) = 0,994589
sin(0,20) = 0,198536     cos(0,20) = 0,979315
sin(0,30) = 0,295232     cos(0,30) = 0,954336
sin(0,40) = 0,388948     cos(0,40) = 0,919904
sin(0,50) = 0,478770     cos(0,50) = 0,876359
sin(0,60) = 0,563823     cos(0,60) = 0,824130
sin(0,70) = 0,643275     cos(0,70) = 0,763730
sin(0,80) = 0,716352     cos(0,80) = 0,695748
sin(0,90) = 0,782337     cos(0,90) = 0,620847
sin(1,00) = 0,840583     cos(1,00) = 0,539756
sin(1,10) = 0,890517     cos(1,10) = 0,453266
sin(1,20) = 0,931645     cos(1,20) = 0,362218
sin(1,30) = 0,963557     cos(1,30) = 0,267498
sin(1,40) = 0,985043     cos(1,40) = 0,169884
sin(1,50) = 0,996736     cos(1,50) = 0,070664
sin(1,60) = 0,998533     cos(1,60) = -0,029189
sin(1,70) = 0,990424     cos(1,70) = -0,128701
sin(1,80) = 0,972493     cos(1,80) = -0,226898
sin(1,90) = 0,944919     cos(1,90) = -0,322822
sin(2,00) = 0,907973     cos(2,00) = -0,415536
sin(2,10) = 0,862016     cos(2,10) = -0,504135
sin(2,20) = 0,807496     cos(2,20) = -0,587751
sin(2,30) = 0,744941     cos(2,30) = -0,665566
sin(2,40) = 0,674960     cos(2,40) = -0,736816
sin(2,50) = 0,598232     cos(2,50) = -0,800800
sin(2,60) = 0,515500     cos(2,60) = -0,856886
sin(2,70) = 0,427192     cos(2,70) = -0,903704
sin(2,80) = 0,334715     cos(2,80) = -0,941512
sin(2,90) = 0,238981     cos(2,90) = -0,969954
sin(3,00) = 0,140927     cos(3,00) = -0,988759
sin(3,10) = 0,041511     cos(3,10) = -0,997749
```

```
Рассчет модуля разности между табулированными и исходными значениями sin и cos

Разность sin = 0,000000          Разность cos = 0,000000
Разность sin = 0,000029          Разность cos = 0,000415
Разность sin = 0,000133          Разность cos = 0,000752
Разность sin = 0,000288          Разность cos = 0,001000
Разность sin = 0,000471          Разность cos = 0,001157
Разность sin = 0,000655          Разность cos = 0,001224
Разность sin = 0,000820          Разность cos = 0,001206
Разность sin = 0,000942          Разность cos = 0,001113
Разность sin = 0,001004          Разность cos = 0,000959
Разность sin = 0,000990          Разность cos = 0,000763
Разность sin = 0,000888          Разность cos = 0,000546
Разность sin = 0,000690          Разность cos = 0,000330
Разность sin = 0,000394          Разность cos = 0,000139
Разность sin = 0,000002          Разность cos = 0,000000
Разность sin = 0,000407          Разность cos = 0,000083
Разность sin = 0,000759          Разность cos = 0,000073
Разность sin = 0,001040          Разность cos = 0,000010
Разность sin = 0,001240          Разность cos = 0,000144
Разность sin = 0,001354          Разность cos = 0,000304
Разность sin = 0,001381          Разность cos = 0,000467
Разность sin = 0,001324          Разность cos = 0,000610
Разность sin = 0,001193          Разность cos = 0,000711
Разность sin = 0,001001          Разность cos = 0,000750
Разность sin = 0,000764          Разность cos = 0,000710
Разность sin = 0,000503          Разность cos = 0,000577
Разность sin = 0,000241          Разность cos = 0,000343
Разность sin = 0,000002          Разность cos = 0,000003
Разность sin = 0,000188          Разность cos = 0,000368
Разность sin = 0,000273          Разность cos = 0,000711
Разность sin = 0,000268          Разность cos = 0,001004
Разность sin = 0,000193          Разность cos = 0,001233
Разность sin = 0,000069          Разность cos = 0,001386
```

```
Сумма квадратов синуса и косинуса

sin(0,00)^2 + cos(0,00)^2 = 1,000000
sin(0,10)^2 + cos(0,10)^2 = 0,999168
sin(0,20)^2 + cos(0,20)^2 = 0,998474
sin(0,30)^2 + cos(0,30)^2 = 0,997919
sin(0,40)^2 + cos(0,40)^2 = 0,997503
sin(0,50)^2 + cos(0,50)^2 = 0,997225
sin(0,60)^2 + cos(0,60)^2 = 0,997086
sin(0,70)^2 + cos(0,70)^2 = 0,997086
sin(0,80)^2 + cos(0,80)^2 = 0,997225
sin(0,90)^2 + cos(0,90)^2 = 0,997502
sin(1,00)^2 + cos(1,00)^2 = 0,997917
sin(1,10)^2 + cos(1,10)^2 = 0,998472
sin(1,20)^2 + cos(1,20)^2 = 0,999165
sin(1,30)^2 + cos(1,30)^2 = 0,999997
sin(1,40)^2 + cos(1,40)^2 = 0,999171
sin(1,50)^2 + cos(1,50)^2 = 0,998476
sin(1,60)^2 + cos(1,60)^2 = 0,997921
sin(1,70)^2 + cos(1,70)^2 = 0,997504
sin(1,80)^2 + cos(1,80)^2 = 0,997226
sin(1,90)^2 + cos(1,90)^2 = 0,997087
sin(2,00)^2 + cos(2,00)^2 = 0,997086
sin(2,10)^2 + cos(2,10)^2 = 0,997224
sin(2,20)^2 + cos(2,20)^2 = 0,997500
sin(2,30)^2 + cos(2,30)^2 = 0,997916
sin(2,40)^2 + cos(2,40)^2 = 0,998470
sin(2,50)^2 + cos(2,50)^2 = 0,999162
sin(2,60)^2 + cos(2,60)^2 = 0,999993
sin(2,70)^2 + cos(2,70)^2 = 0,999173
sin(2,80)^2 + cos(2,80)^2 = 0,998479
sin(2,90)^2 + cos(2,90)^2 = 0,997923
sin(3,00)^2 + cos(3,00)^2 = 0,997506
sin(3,10)^2 + cos(3,10)^2 = 0,997227
```

```
Значение экспонент

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

Значение натурального логарифма

Ln(1) = 0,000000         TabLn(1) = 0,000000
Ln(2) = 0,684939         TabLn(2) = 0,684939
Ln(3) = 1,091556         TabLn(3) = 1,091556
Ln(4) = 1,380907         TabLn(4) = 1,380907
Ln(5) = 1,605475         TabLn(5) = 1,605475
Ln(6) = 1,788942         TabLn(6) = 1,788942
Ln(7) = 1,944016         TabLn(7) = 1,944016
Ln(8) = 2,078299         TabLn(8) = 2,078299
Ln(9) = 2,196703         TabLn(9) = 2,196703
Ln(10) = 2,302585        TabLn(10) = 2,302585
Логарифм от экспоненты():
Exp(0) = 0,000000        TabExp(0) = 0,000000
Exp(1) = 1,000000        TabExp(1) = 1,000000
Exp(2) = 2,000000        TabExp(2) = 2,000000
Exp(3) = 3,000000        TabExp(3) = 3,000000
Exp(4) = 4,000000        TabExp(4) = 4,000000
Exp(5) = 5,000000        TabExp(5) = 5,000000
Exp(6) = 6,000000        TabExp(6) = 6,000000
Exp(7) = 7,000000        TabExp(7) = 7,000000
Exp(8) = 8,000000        TabExp(8) = 8,000000
Exp(9) = 9,000000        TabExp(9) = 9,000000
Exp(10) = 10,000000      TabExp(10) = 10,000000
```