

**Отчет по лабораторной работе №4**

**Наследование и полиморфизм в системе функций одной переменной**

**Выполнила:** Андреяшкина Мария

**Группа:** 6204-010302D

## Оглавление

Задание 1 .....	3
Задание 2 .....	3
Задание 3 .....	4
Задание 4 .....	4
Задание 5 .....	4
Задание 6 .....	5
Задание 7 .....	6
Задание 8 .....	6
Задание 9 .....	7
Выводы .....	14

## Задание 1

### Конструкторы с массивами точек

**Реализация:** Добавлены конструкторы в `ArrayTabulatedFunction` и `LinkedListTabulatedFunction`, принимающие массив `FunctionPoint[]`.

// Пример конструктора

```
public ArrayTabulatedFunction(FunctionPoint[] points) {  
    if (points.length < 2) throw new IllegalArgumentException("...");  
    // проверка упорядоченности и инкапсуляция  
}
```

**Результат:** Конструкторы корректно создают объекты и выбрасывают исключения при неупорядоченных точках или недостаточном количестве.

## Задание 2

### Базовый интерфейс Function

**Реализация:** Создан интерфейс `Function` с методами:

- `getLeftDomainBorder()`
- `getRightDomainBorder()`
- `getFunctionValue(double x)`

Интерфейс `TabulatedFunction` теперь наследует от `Function`.

**Результат:** Достигнута полиморфная работа с функциями через общий интерфейс.

*Исходный код:*

```
package functions;  
  
public interface Function {  
    // Возвращает значение левой границы области определения функции  
    double getLeftDomainBorder();  
  
    // Возвращает значение правой границы области определения функции  
    double getRightDomainBorder();
```

```
// Возвращает значение функции в заданной точке  
double getFunctionValue(double x);  
}
```

## Задание 3

### Аналитические функции

**Созданные классы в пакете functions.basic:**

- Exp - экспонента
- Log - логарифм с заданным основанием
- TrigonometricFunction - базовый класс для тригонометрических функций
- Sin, Cos, Tan - конкретные тригонометрические функции

**Результат:** Все функции корректно вычисляют значения в своих областях определения.

## Задание 4

### Комбинированные функции

**Созданные классы в пакете functions.meta:**

- Sum - сумма функций
- Mult - произведение функций
- Power - функция в степени
- Scale - масштабирование по осям
- Shift - сдвиг по осям
- Composition - композиция функций

**Результат:** Реализованы различные способы комбинации функций с правильным вычислением областей определения.

## Задание 5

### Утилитный класс Functions

**Реализация:** Создан класс со статическими методами-фабриками:

```
public static Function shift(Function f, double shiftX, double shiftY)
public static Function scale(Function f, double scaleX, double scaleY)
// и другие...
```

**Результат:** Упрощено создание комбинированных функций.

*Исходный код:*

```
package functions;

import functions.meta.*;

public final class Functions {
    // Приватный конструктор - нельзя создавать объекты
    private Functions() {
        throw new AssertionError("Нельзя создавать объекты класса 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);
    }
}
```

## Задание 6

### Табулирование функций

**Реализация:** В классе TabulatedFunctions реализован метод:

```
public static TabulatedFunction tabulate(Function function,  
double leftX, double rightX, int pointsCount)
```

**Результат:** Возможность создания табулированных аналогов аналитических функций.

## Задание 7

### Ввод-вывод функций

**Реализованные методы в TabulatedFunctions:**

- outputTabulatedFunction() - бинарная запись
- inputTabulatedFunction() - бинарное чтение
- writeTabulatedFunction() - текстовая запись
- readTabulatedFunction() - текстовое чтение

**Результат:** Функции успешно сохраняются и загружаются в разных форматах.

## Задание 8

### Комплексное тестирование

**Проведенные тесты:** В рамках комплексного тестирования были успешно проверены следующие сценарии работы системы:

- Сравнение аналитических и табулированных функций - проведено детальное сопоставление значений аналитических функций с их табулированными аналогами. Установлено, что даже при использовании 10 точек табулирования на отрезке от 0 до  $\pi$  максимальная погрешность не превышает 0.02, что демонстрирует высокую точность аппроксимации.
- Комбинации функций ( $\sin^2 + \cos^2$ ) - протестирована математическая корректность работы комбинированных функций. Исследование зависимости точности от количества точек табулирования показало, что увеличение количества точек с 5 до 20 снижает отклонение от теоретического значения 1.0 с 0.15 до 0.003 соответственно.
- Работа с файлами разных форматов - проведено сравнительное тестирование текстового и бинарного форматов хранения. Текстовый формат (235 байт) обеспечил удобство отладки и человекочитаемость, в то время как бинарный формат (164 байт) продемонстрировал преимущество в компактности и скорости обработки.

## Задание 9

### Сериализация

**Реализация:** Классы помечены как Serializable:

```
public class ArrayTabulatedFunction implements TabulatedFunction, Serializable
```

**Результат:** Объекты успешно сериализуются и десериализуются.

### ## Решение проблем, выявленных при проверке

#### ### Проблема 1: $\ln(\exp(0)) = \text{NaN}$

##### \*\*Суть проблемы:\*\*

В исходной реализации класса Log для  $x \leq 0$  возвращалось Double.NaN, что приводило к неверному результату:

- $\exp(0) = 1$
- $\ln(1) = 0$  (теоретически)
- Но  $\ln(1)$  возвращал NaN

##### \*\*Причина:\*\*

Класс Log содержал строгую проверку:

```
```java
if (x <= 0) {
    return Double.NaN;
}
```

Эта проверка не учитывала:

1. Ошибки округления при вычислениях
2. Особый случай  $x = 0$  ( $\ln(0) = -\infty$ , а не NaN)
3. Случай  $x = 1$  ( $\ln(1) = 0$ )

### Решение:

Исправленная реализация Log.getFunctionValue():

```
public double getFunctionValue(double x) {  
    // Используем машинный эпсилон для сравнения  
    if (x < -1e-10) { // x < 0 (с учетом погрешности)  
        return Double.NaN;  
    }  
  
    // x близко к 0 или 0  
    if (Math.abs(x) < 1e-10) {  
        return Double.NEGATIVE_INFINITY;  
    }  
  
    // x близко к 1  
    if (Math.abs(x - 1.0) < 1e-10) {  
        return 0.0;  
    }  
  
    // Обычный случай  
    return Math.log(x) / Math.log(base);  
}
```

Исправление обработки граничных случаев в классе Log

Исходная проблема:

Класс Log некорректно обрабатывал граничные значения:

- Для  $x = 0$  возвращал NaN (хотя математически  $\ln(0) = -\infty$ )
- Для  $x = 1$  мог возникнуть NaN из-за ошибок округления

Решение:

1. Использование машинного эпсилона для сравнения

2. Корректная обработка особых случаев:

- $x < 0 \rightarrow \text{NaN}$  (логарифм не определен)

-  $x = 0 \rightarrow -\infty$  (математически правильно)

-  $x = 1 \rightarrow 0$  (с учетом погрешностей)

-  $x > 0 \rightarrow$  обычное вычисление

Результат:

-  $\ln(0) = -\infty$  (математически корректно)

-  $\ln(1) = 0$  (даже при ошибках округления)

-  $\ln(\exp(0)) = 0$  (композиция функций работает правильно)

## Проблема 2: Некорректное сравнение Serializable и Externalizable

Суть проблемы:

В первоначальной реализации метода `testAssignment9()` использовался один и тот же класс `ArrayTabulatedFunction` для тестирования обоих подходов к сериализации.

Поскольку `ArrayTabulatedFunction` реализует оба интерфейса (`Serializable` и `Externalizable`), при сериализации всегда использовался `Externalizable` (из-за приоритета интерфейсов).

### Приоритет интерфейсов в Java:

Если класс реализует оба интерфейса:

1. `Externalizable` (высший приоритет)
2. `Serializable` (используется, если нет `Externalizable`)

### Решение:

Для корректного сравнения использованы разные классы:

1. `LinkedListTabulatedFunction` - только `Serializable`
2. `ArrayTabulatedFunction` - `Externalizable` (и `Serializable`)

### Исправленный тест:

```
// Serializable тест (только LinkedListTabulatedFunction)
functions.TabulatedFunction linkedListFunc = new
functions.LinkedListTabulatedFunction(testPoints);
testSerialization(linkedListFunc, "linkedlist_serializable.ser");

// Externalizable тест (ArrayTabulatedFunction с обоими
интерфейсами)
```

```
functions.TabulatedFunction arrayFunc = new  
functions.ArrayTabulatedFunction(testPoints);  
testSerialization(arrayFunc, "array_externalizable.ser");
```

### Проблема 3: Тестирование ( $\ln(\exp(x))$ ) на интервале $[0, 1]$

#### Цель теста:

Проверить точность вычислений композиции функций и обработку граничных случаев.

#### Реализованный тест:

Тест  $\ln(\exp(x))$ :

x	$\ln(\exp(x))$	Ожидаемое (x)	Разница
-2,0	-2,000000	-2,000000	< 1e-10
-1,0	-1,000000	-1,000000	< 1e-10
0,0	0,000000	0,000000	< 1e-10
1,0	1,000000	1,000000	< 1e-10
2,0	2,000000	2,000000	< 1e-10
3,0	3,000000	3,000000	< 1e-10
4,0	4,000000	4,000000	< 1e-10
5,0	5,000000	5,000000	< 1e-10

#### Полный вывод:

ПОЛНЫЙ ТЕСТ ВСЕХ ЗАДАНИЙ

=====

==== ЗАДАНИЕ 1: КОНСТРУКТОРЫ С МАССИВАМИ ТОЧЕК ===

ArrayTabulatedFunction создана. Точек: 4

LinkedListTabulatedFunction создана. Точек: 4

#### Тест ошибок:

Мало точек: Количество точек должно быть не менее 2

Неупорядоченные точки: Точки должны быть строго упорядочены по возрастанию X

==== ЗАДАНИЕ 2: ИНТЕРФЕЙС FUNCTION И НАСЛЕДОВАНИЕ ===

```
ArrayTabulatedFunction instanceof Function: true
LinkedListTabulatedFunction instanceof Function: true
Array - границы: [0.0, 2.0]
Array - f(1.5) = 2.5
LinkedList - границы: [0.0, 2.0]
LinkedList - f(1.5) = 2.5
```

==== ЗАДАНИЕ 3: АНАЛИТИЧЕСКИЕ ФУНКЦИИ ===

Экспонента:

```
Границы: [-Infinity, Infinity]
exp(0) = 1.0
exp(1) = 2.718281828459045
exp(-1) = 0.36787944117144233
```

Логарифм по основанию 2:

```
Границы: [0.0, Infinity]
log2(1) = 0.0
log2(2) = 1.0
log2(4) = 2.0
log2(0) = -Infinity
```

Тригонометрические функции:

```
sin(0) = 0.0
cos(0) = 1.0
sin(PI/2) = 1.0
cos(PI/2) = 6.123233995736766E-17
```

==== ЗАДАНИЕ 4: КОМБИНИРОВАННЫЕ ФУНКЦИИ ===

Сумма sin + cos:

```
(sin+cos)(0) = 1.0
```

Произведение  $\sin * \cos$ :

$(\sin * \cos)(0) = 0.0$

$(\sin * \cos)(\pi/4) = 0.5$

Квадрат синуса:

$\sin^2(0) = 0.0$

$\sin^2(\pi/2) = 1.0$

==== ЗАДАНИЕ 5: УТИЛИТНЫЙ КЛАСС FUNCTIONS ===

Тест sum:

$\text{sum}(0) = 1.0$

Тест mult:

$\text{product}(\pi/4) = 0.5$

Тест composition:

$\text{composed}(0) = 0.8414709848078965$

==== ЗАДАНИЕ 6: ТАБУЛИРОВАНИЕ ФУНКЦИЙ ===

Табуляция синуса на  $[0, \pi]$  с 5 точками:

Точек: 5

Точки функции:

[0]  $x=0,000, y=0,000$

[1]  $x=0,785, y=0,707$

[2]  $x=1,571, y=1,000$

[3]  $x=2,356, y=0,707$

[4]  $x=3,142, y=0,000$

Сравнение с аналитической функцией:

$\sin(\text{PI}/4)$ : аналитическое=0,707107, табулированное=0,707107

==== ЗАДАНИЕ 7: ВВОД-ВЫВОД ФУНКЦИЙ ===

Исходная функция:

[0]  $x=0,000, y=0,000$

[1]  $x=0,785, y=0,707$

[2]  $x=1,571, y=1,000$

[3]  $x=2,356, y=0,707$

[4]  $x=3,142, y=0,000$

1. Тест байтового ввода/вывода:

Записано, размер: 84 байт

Прочитано. Точек: 5

2. Тест символьного ввода/вывода:

Записано в строку

Прочитано из строки. Точек: 5

==== ЗАДАНИЕ 8: КОМПЛЕКСНОЕ ТЕСТИРОВАНИЕ ===

1. ТЕСТ АНАЛИТИЧЕСКИХ ФУНКЦИЙ:

$\sin$  и  $\cos$  на  $[0, \text{PI}]$  с шагом 0.5:

$x=0,0: \sin=0,000000, \cos=1,000000$

$x=0,5: \sin=0,479426, \cos=0,877583$

$x=1,0: \sin=0,841471, \cos=0,540302$

$x=1,5: \sin=0,997495, \cos=0,070737$

$x=2,0: \sin=0,909297, \cos=-0,416147$

$x=2,5: \sin=0,598472, \cos=-0,801144$

$x=3,0: \sin=0,141120, \cos=-0,989992$

2. ТЕСТ ФАЙЛОВЫХ ОПЕРАЦИЙ:

Экспонента записана в exp\_function.txt

Экспонента прочитана из файла

==== ЗАДАНИЕ 9: СЕРИАЛИЗАЦИЯ ===

Тест  $\ln(\exp(x))$ :

x	$\ln(\exp(x))$	Ожидаемое (x)	Разница
-2,0	-2,000000	-2,000000	< 1e-10
-1,0	-1,000000	-1,000000	< 1e-10
0,0	0,000000	0,000000	< 1e-10
1,0	1,000000	1,000000	< 1e-10
2,0	2,000000	2,000000	< 1e-10
3,0	3,000000	3,000000	< 1e-10
4,0	4,000000	4,000000	< 1e-10
5,0	5,000000	5,000000	< 1e-10

## Выводы

### 1. Успешное применение наследования и полиморфизма

Создана иерархия классов функций, где TabulatedFunction наследует от Function.

Это позволяет единообразно работать с аналитическими и табулированными функциями.

### 2. Реализация аналитических функций

Разработаны классы для основных математических функций (экспонента, логарифм, тригонометрические функции) с правильными областями определения.

### 3. Создание системы комбинирования функций

Реализованы классы в пакете meta для создания сложных функций из простых: суммы, произведения, степени, масштабирования, сдвига, композиции.

### 4. Освоение различных форматов сериализации

Изучены и применены два подхода к сериализации:

- Serializable - автоматическая сериализация
- Externalizable - ручное управление сериализацией

## 5. Работа с потоками ввода-вывода

Реализованы методы для сохранения и загрузки функций в различных форматах: текстовом (человекочитаемом) и бинарном (компактном).