

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	18

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 π 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:

- $\ln(0) = -\text{Infinity}$  (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:

```
--- ЗАДАНИЕ 1: Конструкторы с массивами точек ---
ArrayTabulatedFunction создана. Точек: 3
LinkedListTabulatedFunction создана. Точек: 3

--- ЗАДАНИЕ 2: Интерфейс Function ---
Функция реализует интерфейс Function: true
Границы: [0.0, 2.0]
f(1.5) = 2.5

--- ЗАДАНИЕ 3: Аналитические функции ---
Экспонента: $\exp(0) = 1.0$
```

Логарифм по основанию 2:  $\log_2(8) = 3.0$

Тригонометрические функции:

$\sin(0) = 0.0$

$\cos(0) = 1.0$

$\tan(0) = 0.0$

--- ЗАДАНИЕ 4: Комбинированные функции ---

Сумма  $\sin + \cos$  в точке 0: 1.0

Произведение  $\sin * \cos$  в точке  $\pi/4$ : 0.5

Квадрат синуса в точке  $\pi/2$ : 1.0

--- ЗАДАНИЕ 5: Утилитный класс Functions ---

`Functions.sum(sin,cos)` в 0: 1.0

`Functions.mult(sin,cos)` в  $\pi/4$ : 0.5

`Functions.power(sin,2)` в  $\pi/2$ : 1.0

--- ЗАДАНИЕ 6: Табулирование функций ---

Табулированный синус с 5 точками:

(0,00, 0,0000)

(0,79, 0,7071)

(1,57, 1,0000)

(2,36, 0,7071)

(3,14, 0,0000)

--- ЗАДАНИЕ 7: Ввод-вывод функций ---

Байтовый ввод-вывод: успешно

Символьный ввод-вывод: успешно

--- ЗАДАНИЕ 8: Комплексное тестирование ---

=== Часть 1: Основные функции ===

sin и cos на  $[0, \pi]$  с шагом 0.1:

x=0,0:	sin=0,000000,	cos=1,000000
x=0,1:	sin=0,099833,	cos=0,995004
x=0,2:	sin=0,198669,	cos=0,980067
x=0,3:	sin=0,295520,	cos=0,955336
x=0,4:	sin=0,389418,	cos=0,921061
x=0,5:	sin=0,479426,	cos=0,877583
x=0,6:	sin=0,564642,	cos=0,825336
x=0,7:	sin=0,644218,	cos=0,764842
x=0,8:	sin=0,717356,	cos=0,696707
x=0,9:	sin=0,783327,	cos=0,621610
x=1,0:	sin=0,841471,	cos=0,540302
x=1,1:	sin=0,891207,	cos=0,453596
x=1,2:	sin=0,932039,	cos=0,362358
x=1,3:	sin=0,963558,	cos=0,267499
x=1,4:	sin=0,985450,	cos=0,169967
x=1,5:	sin=0,997495,	cos=0,070737
x=1,6:	sin=0,999574,	cos=-0,029200
x=1,7:	sin=0,991665,	cos=-0,128844
x=1,8:	sin=0,973848,	cos=-0,227202
x=1,9:	sin=0,946300,	cos=-0,323290
x=2,0:	sin=0,909297,	cos=-0,416147
x=2,1:	sin=0,863209,	cos=-0,504846
x=2,2:	sin=0,808496,	cos=-0,588501
x=2,3:	sin=0,745705,	cos=-0,666276
x=2,4:	sin=0,675463,	cos=-0,737394
x=2,5:	sin=0,598472,	cos=-0,801144
x=2,6:	sin=0,515501,	cos=-0,856889
x=2,7:	sin=0,427380,	cos=-0,904072
x=2,8:	sin=0,334988,	cos=-0,942222

x=2,9: sin=0,239249, cos=-0,970958

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

x=3,1: sin=0,041581, cos=-0,999135

=== Часть 2: Табулированные аналоги ===

Сравнение исходных и табулированных:

x=0,0: sin: 0,000000 vs 0,000000, cos: 1,000000 vs 1,000000

x=0,1: sin: 0,099833 vs 0,097982, cos: 0,995004 vs 0,982723

x=0,2: sin: 0,198669 vs 0,195963, cos: 0,980067 vs 0,965446

x=0,3: sin: 0,295520 vs 0,293945, cos: 0,955336 vs 0,948170

x=0,4: sin: 0,389418 vs 0,385907, cos: 0,921061 vs 0,914355

x=0,5: sin: 0,479426 vs 0,472070, cos: 0,877583 vs 0,864608

x=0,6: sin: 0,564642 vs 0,558234, cos: 0,825336 vs 0,814862

x=0,7: sin: 0,644218 vs 0,643982, cos: 0,764842 vs 0,764620

x=0,8: sin: 0,717356 vs 0,707935, cos: 0,696707 vs 0,688404

x=0,9: sin: 0,783327 vs 0,771888, cos: 0,621610 vs 0,612188

x=1,0: sin: 0,841471 vs 0,835841, cos: 0,540302 vs 0,535972

x=1,1: sin: 0,891207 vs 0,883993, cos: 0,453596 vs 0,450633

x=1,2: sin: 0,932039 vs 0,918022, cos: 0,362358 vs 0,357141

x=1,3: sin: 0,963558 vs 0,952051, cos: 0,267499 vs 0,263648

x=1,4: sin: 0,985450 vs 0,984808, cos: 0,169967 vs 0,169931

x=1,5: sin: 0,997495 vs 0,984808, cos: 0,070737 vs 0,070437

x=1,6: sin: 0,999574 vs 0,984808, cos: -0,029200 vs -0,029056

x=1,7: sin: 0,991665 vs 0,984808, cos: -0,128844 vs -0,128549

x=1,8: sin: 0,973848 vs 0,966204, cos: -0,227202 vs -0,224761

x=1,9: sin: 0,946300 vs 0,932175, cos: -0,323290 vs -0,318254

x=2,0: sin: 0,909297 vs 0,898147, cos: -0,416147 vs -0,411747

x=2,1: sin: 0,863209 vs 0,862441, cos: -0,504846 vs -0,504272

x=2,2: sin: 0,808496 vs 0,798488, cos: -0,588501 vs -0,580488

x=2,3: sin: 0,745705 vs 0,734535, cos: -0,666276 vs -0,656704

x=2,4: sin: 0,675463 vs 0,670582, cos: -0,737394 vs -0,732920  
x=2,5: sin: 0,598472 vs 0,594072, cos: -0,801144 vs -0,794171  
x=2,6: sin: 0,515501 vs 0,507908, cos: -0,856889 vs -0,843917  
x=2,7: sin: 0,427380 vs 0,421745, cos: -0,904072 vs -0,893664  
x=2,8: sin: 0,334988 vs 0,334698, cos: -0,942222 vs -0,940984  
x=2,9: sin: 0,239249 vs 0,236716, cos: -0,970958 vs -0,958261  
x=3,0: sin: 0,141120 vs 0,138735, cos: -0,989992 vs -0,975537  
x=3,1: sin: 0,041581 vs 0,040753, cos: -0,999135 vs -0,992814

=== Часть 3: Сумма квадратов ===

sin<sup>2</sup> + cos<sup>2</sup> (должно быть ? 1):

x=0,0: 1,000000, отклонение от 1: 0,000000  
x=0,1: 0,975345, отклонение от 1: 0,024655  
x=0,2: 0,970488, отклонение от 1: 0,029512  
x=0,3: 0,985429, отклонение от 1: 0,014571  
x=0,4: 0,984968, отклонение от 1: 0,015032  
x=0,5: 0,970398, отклонение от 1: 0,029602  
x=0,6: 0,975624, отклонение от 1: 0,024376  
x=0,7: 0,999358, отклонение от 1: 0,000642  
x=0,8: 0,975073, отклонение от 1: 0,024927  
x=0,9: 0,970586, отклонение от 1: 0,029414  
x=1,0: 0,985897, отклонение от 1: 0,014103  
x=1,1: 0,984515, отклонение от 1: 0,015485  
x=1,2: 0,970314, отклонение от 1: 0,029686  
x=1,3: 0,975910, отклонение от 1: 0,024090  
x=1,4: 0,998723, отклонение от 1: 0,001277  
x=1,5: 0,974808, отклонение от 1: 0,025192  
x=1,6: 0,970691, отклонение от 1: 0,029309  
x=1,7: 0,986371, отклонение от 1: 0,013629  
x=1,8: 0,984068, отклонение от 1: 0,015932

x=1,9: 0,970237, отклонение от 1: 0,029763  
x=2,0: 0,976203, отклонение от 1: 0,023797  
x=2,1: 0,998094, отклонение от 1: 0,001906  
x=2,2: 0,974549, отклонение от 1: 0,025451  
x=2,3: 0,970802, отклонение от 1: 0,029198  
x=2,4: 0,986852, отклонение от 1: 0,013148  
x=2,5: 0,983628, отклонение от 1: 0,016372  
x=2,6: 0,970167, отклонение от 1: 0,029833  
x=2,7: 0,976503, отклонение от 1: 0,023497  
x=2,8: 0,997473, отклонение от 1: 0,002527  
x=2,9: 0,974298, отклонение от 1: 0,025702  
x=3,0: 0,970920, отклонение от 1: 0,029080  
x=3,1: 0,987341, отклонение от 1: 0,012659

=== Часть 4: Экспонента (текстовый файл) ===

Записано в exp\_text.txt

Сравнение:

x=0: 1,000000 vs 1,000000  
x=1: 2,718282 vs 2,718282  
x=2: 7,389056 vs 7,389056  
x=3: 20,085537 vs 20,085537  
x=4: 54,598150 vs 54,598150  
x=5: 148,413159 vs 148,413159  
x=6: 403,428793 vs 403,428793  
x=7: 1096,633158 vs 1096,633158  
x=8: 2980,957987 vs 2980,957987  
x=9: 8103,083928 vs 8103,083928  
x=10: 22026,465795 vs 22026,465795

=== Часть 5: Натуральный логарифм (байтовый файл) ===

Записано в ln\_binary.bin

Сравнение:

x=1: 0,000000 vs 0,000000  
x=2: 0,693147 vs 0,693147  
x=3: 1,098612 vs 1,098612  
x=4: 1,386294 vs 1,386294  
x=5: 1,609438 vs 1,609438  
x=6: 1,791759 vs 1,791759  
x=7: 1,945910 vs 1,945910  
x=8: 2,079442 vs 2,079442  
x=9: 2,197225 vs 2,197225  
x=10: 2,302585 vs 2,302585

--- ЗАДАНИЕ 9: Сериализация ---

=== ЗАДАНИЕ 9: СЕРИАЛИЗАЦИЯ ===

$\ln(\exp(x)) = x$  (теоретически):

x=0:  $\ln(\exp(x)) = 0,000000$ , ожидается 0,000000  
x=1:  $\ln(\exp(x)) = 1,000000$ , ожидается 1,000000  
x=2:  $\ln(\exp(x)) = 2,000000$ , ожидается 2,000000  
x=3:  $\ln(\exp(x)) = 3,000000$ , ожидается 3,000000  
x=4:  $\ln(\exp(x)) = 4,000000$ , ожидается 4,000000  
x=5:  $\ln(\exp(x)) = 5,000000$ , ожидается 5,000000  
x=6:  $\ln(\exp(x)) = 6,000000$ , ожидается 6,000000  
x=7:  $\ln(\exp(x)) = 7,000000$ , ожидается 7,000000  
x=8:  $\ln(\exp(x)) = 8,000000$ , ожидается 8,000000  
x=9:  $\ln(\exp(x)) = 9,000000$ , ожидается 9,000000  
x=10:  $\ln(\exp(x)) = 10,000000$ , ожидается 10,000000

--- Serializable сериализация ---

Записано в function\_serializable.ser

Сравнение после Serializable:

x=0: 0,000000 vs 0,000000

x=1: 1,000000 vs 1,000000

x=2: 2,000000 vs 2,000000

x=3: 3,000000 vs 3,000000

x=4: 4,000000 vs 4,000000

x=5: 5,000000 vs 5,000000

x=6: 6,000000 vs 6,000000

x=7: 7,000000 vs 7,000000

x=8: 8,000000 vs 8,000000

x=9: 9,000000 vs 9,000000

x=10: 10,000000 vs 10,000000

--- Externalizable сериализация ---

Записано в function\_externalizable.ser

Сравнение после Externalizable:

x=0: 0,000000 vs 0,000000

x=1: 1,000000 vs 1,000000

x=2: 2,000000 vs 2,000000

x=3: 3,000000 vs 3,000000

x=4: 4,000000 vs 4,000000

x=5: 5,000000 vs 5,000000

x=6: 6,000000 vs 6,000000

x=7: 7,000000 vs 7,000000

x=8: 8,000000 vs 8,000000

x=9: 9,000000 vs 9,000000

x=10: 10,000000 vs 10,000000

Размеры файлов:

Serializable: 236 байт

Externalizable: 236 байт

## 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).