Отчет по лабораторной работе №4

Наследование и полиморфизм в системе функций одной переменной

Выполнила: Андреяшкина Мария
Группа: 6204-010302D

Оглавление

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)) = NaN

****Суть проблемы:****

В исходной реализации класса Log для x ≤ 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) = -∞, 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) = -∞)

- Для x = 1 мог возникнуть NaN из-за ошибок округления

Решение:

1. Использование машинного эпсилона для сравнения

2. Корректная обработка особых случаев:

  - x < 0 → NaN (логарифм не определен)

- x = 0 → -∞ (математически правильно)

- x = 1 → 0 (с учетом погрешностей)

- x > 0 → обычное вычисление

Результат:

- ln(0) = -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:

```
System.out.println("Тестирование ln(exp(x)) на [0, 1]:");

System.out.println("x\tТеоретическое\tФактическое\tРазница");

for (double x = 0; x <= 1.0; x += 0.1) {

    double theoretical = x;

    double actual = logOfExp.getFunctionValue(x);

    double diff = Math.abs(theoretical - actual);

    System.out.printf("%.1f\t%.6f\t%.6f\t%.10f%n",

        x, theoretical, actual, diff);

}
```

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).