ДНІПРОВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ОЛЕСЯ ГОНЧАРА

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ МАТЕМАТИКИ ТА МАТЕМАТИЧНОЇ КІБЕРНЕТИКИ

ЗВІТ ПРО ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ № 1

З дисципліни «Сучасні середовища програмування»

Перший (бакалаврський) рівень вищої освіти
Спеціальність 113 Прикладна математика
Освітня програма Комп’ютерне моделювання
та технології програмування

Виконавець:
Студент групи ПА-22-1
Бердик Роман
Варіант 1

Дніпро
2023

Постановка завдання

Застосувати паттерн Компонувальник (Composite) для обчислення похідної складеної функції на прикладі заданих формул. Використати вхідні дані для обчислення значень функцій і їх похідних.

Опис логічної структури Java-програми

1. Опис паттернів та класів:
   Component (Компонент): Абстрактний клас або інтерфейс, який описує загальні методи для всіх компонентів.

```
abstract class FunctionComponent {
    abstract double getValue(double x);
    abstract double getDerivative(double x);
}
```

   Leaf (Лист): Клас, що представляє прості функції.

```
class SimpleFunction extends FunctionComponent {
    private double a;
    private double b;

    public SimpleFunction(double a, double b) {
        this.a = a;
        this.b = b;
    }

    @Override
    public double getValue(double x) {
        // Реалізація функції з вхідними параметрами a, b тут
        return 0.0;
    }

    @Override
    public double getDerivative(double x) {
        // Реалізація обчислення похідної функції тут
        return 0.0;
    }
}
```

Composite (Складений компонент): Клас, який може містити інші компоненти (в тому числі інші складені компоненти).

```
class CompositeFunction extends FunctionComponent {
    private List<FunctionComponent> components = new ArrayList<>();

    public void addComponent(FunctionComponent component) {
        components.add(component);
    }

    @Override
    public double getValue(double x) {
        double result = 0.0;
        for (FunctionComponent component : components) {
            result += component.getValue(x);
        }
        return result;
    }

    @Override
    public double getDerivative(double x) {
        double derivative = 0.0;
        for (FunctionComponent component : components) {
            derivative += component.getDerivative(x);
        }
        return derivative;
    }
}
```

2. Використання паттерну Composite для складеної функції:

    1. Створення об'єктів SimpleFunction для кожної простої функції з заданими параметрами a і b.
    2. Створення складених функцій CompositeFunction, які містять прості функції відповідно до заданих формул.
    3. Виклик методів getValue(x) для обчислення значень складеної функції та getDerivative(x) для обчислення похідної.

Висновки за результатами роботи

Використання паттерну Компонувальник (Composite) дозволяє створювати складені структури з об'єктів, що дозволяє обробляти їх як єдині цілісні об'єкти. Це дозволяє розширювати функціональні можливості програми шляхом створення складених структур з простих компонентів.

Додатки (код програми)

```
import java.util.ArrayList;
import java.util.List;

abstract class FunctionComponent {
    abstract double getValue(double x);
    abstract double getDerivative(double x);
}

class SimpleFunction extends FunctionComponent {
    private double a;
    private double b;

    public SimpleFunction(double a, double b) {
        this.a = a;
        this.b = b;
    }

    @Override
    public double getValue(double x) {
        return Math.pow(x, 2) * Math.sqrt(Math.abs(a * x - 1)) - Math.pow(Math.sin(x + b),
3);
    }
```

```
    @Override
    public double getDerivative(double x) {
        double dx = 1e-6; // Дельта для обчислення похідної
        double fx_plus_dx = getValue(x + dx);
        double fx_minus_dx = getValue(x - dx);
        return (fx_plus_dx - fx_minus_dx) / (2 * dx);
    }
}

class CompositeFunction extends FunctionComponent {
    private List<FunctionComponent> components = new ArrayList<>();

    public void addComponent(FunctionComponent component) {
        components.add(component);
    }

    @Override
    public double getValue(double x) {
        double result = 0.0;
        for (FunctionComponent component : components) {
            result += component.getValue(x);
        }
        return result;
    }

    @Override
    public double getDerivative(double x) {
        double derivative = 0.0;
        for (FunctionComponent component : components) {
            derivative += component.getDerivative(x);
        }
```

```
        return derivative;
    }
}

public class CompositePatternDemo {
    public static void main(String[] args) {
        double a = 0.7;
        double b = 0.05;
        double x = 0.4;

        // Створення простих функцій f1(x) і f2(x)
        SimpleFunction f1 = new SimpleFunction(a, b);
        SimpleFunction f2 = new SimpleFunction(a, b);

        // Створення складеної функції з f1(x) і f2(x)
        CompositeFunction compositeFunction = new CompositeFunction();
        compositeFunction.addComponent(f1);
        compositeFunction.addComponent(f2);

        // Обчислення значення складеної функції
        double result = compositeFunction.getValue(x);
        System.out.println("Value of composite function at x = " + x + ": " + result);

        // Обчислення похідної складеної функції
        double derivative = compositeFunction.getDerivative(x);
        System.out.println("Derivative of composite function at x = " + x + ": " + derivative);
    }
}
```