Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №8

Технології розробки програмного забезпечення

ШАБЛОНИ «COMPOSITE», «FLYWEIGHT», «INTERPRETER», «VISITOR»

Виконала
студентка групи ІА-24
Тильна Марія Сергіївна

Перевірив:
Мягкий М.Ю.

Київ 2024

Зміст

Тема: ШАБЛОНИ «COMPOSITE», «FLYWEIGHT», «INTERPRETER», «VISITOR»

Короткі теоретичні відомості

Шаблони проектування "Composite", "Flyweight", "Interpreter" і "Visitor" є структурними або поведінковими патернами, які допомагають вирішувати різні типи проблем у розробці програмного забезпечення.

Composite (Композиція) є структурним шаблоном проектування, який дозволяє об'єднати об'єкти в дерево так, що клієнти можуть працювати з окремими об'єктами та їх композиціями однаково. Це дозволяє будувати складні структури, де компоненти можуть бути як простими, так і складеними, і забезпечує зручний інтерфейс для їх взаємодії.

Flyweight (Летючий вагон) є структурним шаблоном проектування, що дозволяє економити ресурси за рахунок спільного використання великої кількості об'єктів, що мають однакові дані або поведінку. Flyweight зберігає спільні властивості об'єктів, а специфічні дані передає через контекст, дозволяючи значно зменшити використання пам'яті.

Interpreter (Інтерпретатор) є поведінковим шаблоном проектування, який визначає граматику певної мови та надає спосіб інтерпретації виразів цієї мови. Цей шаблон зазвичай використовується для створення інтерпретаторів мов або для реалізації специфічних бізнес-правил, які можуть бути представлені у вигляді виразів чи синтаксичних дерев.

Visitor (Візитер) є поведінковим шаблоном проектування, що дозволяє додавати нову поведінку до об'єктів без зміни їх класів. Візитер використовується для обробки об'єктів різних класів за допомогою однієї операції, що дає можливість додавати нові операції до структури об'єктів, не змінюючи їх. Це підвищує гнучкість і дозволяє легко додавати нові функціональності до існуючих класів.

Хід роботи

..11 Web crawler (proxy, chain of responsibility, memento, template method, composite, p2p)

Веб-сканер повинен вміти розпізнавати структуру сторінок сайту, переходити за посиланнями, збирати необхідну інформацію про зазначений термін, видаляти не семантичні одиниці (рекламу, об'єкти javascript і т.д.), зберігати знайдені дані у вигляді структурованого набору html файлів вести статистику відвіданих сайтів і метадані.

У проєкті веб-сканера я використала шаблон Composite для того щоб організувати статистичні дані у вигляді ієрархії. Метою використання цього шаблону було створення гнучкої структури для зберігання статистичних даних, що дозволяє легко розширювати систему та додавати нові компоненти статистики без необхідності змінювати основний код. Шаблон "Composite" дозволяє об'єднувати об'єкти в деревоподібні структури для представлення ієрархій "частина-ціле". Це дозволяє клієнтам обробляти окремі об'єкти і композиції об'єктів однаково.

Переваги шаблона Composite:

Масштабованість: Легко розширювати систему шляхом додавання нових типів статистичних даних.

Повторне використання: Компоненти можуть бути повторно використані в різних частинах системи без зміни їх логіки.

Зменшення складності: Спрощує управління складними структурами даних шляхом їх об'єднання в єдину ієрархію.

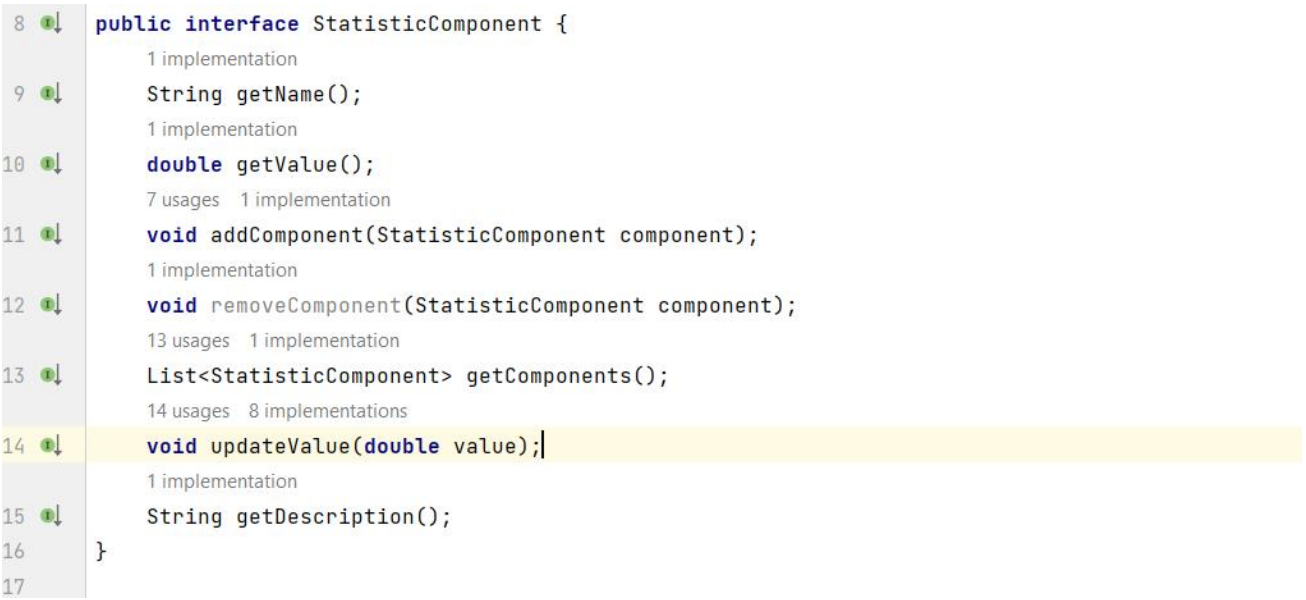


Рисунок 1 - Код інтерфейсу StatisticComponent

Інтерфейс StatisticComponent є базовим інтерфейсом для всіх компонентів статистики у вашому проекті. Він визначає контракт, який повинні реалізувати всі статистичні компоненти. Ось основні методи та їх призначення:

Основні методи StatisticComponent

getName() повертає назву компонента статистики. Це дозволяє ідентифікувати компонент в ієрархії статистичних даних.

getValue() повертає значення статистики для компонента.

addComponent(StatisticComponent component) додає дочірній компонент до поточного компонента.

removeComponent(StatisticComponent component) видаляє дочірній компонент з поточного компонента.

getComponents() повертає список всіх дочірніх компонентів.**updateValue(double value) о**новлює значення статистики для компонента.

getDescription() повертає повний опис компонента, включаючи його назву, значення та підкомпоненти.

```
public abstract class BaseStatistic implements StatisticComponent {
    protected String name;
    protected double value;
    7 usages
    protected List<StatisticComponent> components;
    8 usages
    public BaseStatistic(String name) {
        this.name = name;
        this.value = 0.0;
        this.components = new ArrayList<>();
    }
    @Override
    public String getName() { return name; }
    @Override
    public double getValue() { return value; }
    7 usages
    @Override
    public void addComponent(StatisticComponent component) {
        if (!components.contains(component)) {
            components.add(component);
        }
    }
    @Override
    public void removeComponent(StatisticComponent component) { components.remove(component); }
    13 usages
    @Override
    public List<StatisticComponent> getComponents() { return new ArrayList<>(components); }
    14 usages  7 overrides
    @Override
    public void updateValue(double value) { this.value = value; }

    @Override
    public String getDescription() {
        StringBuilder description = new StringBuilder();
        description.append(name).append(": ").append(value);
        if (!components.isEmpty()) {
            description.append("\nSub-components:");
            for (StatisticComponent component : components) {
                description.append("\n  ").append(component.getDescription().replace( target: "\n",  replacement: "\n  "));
            }
        }
        return description.toString();
    }
}
```

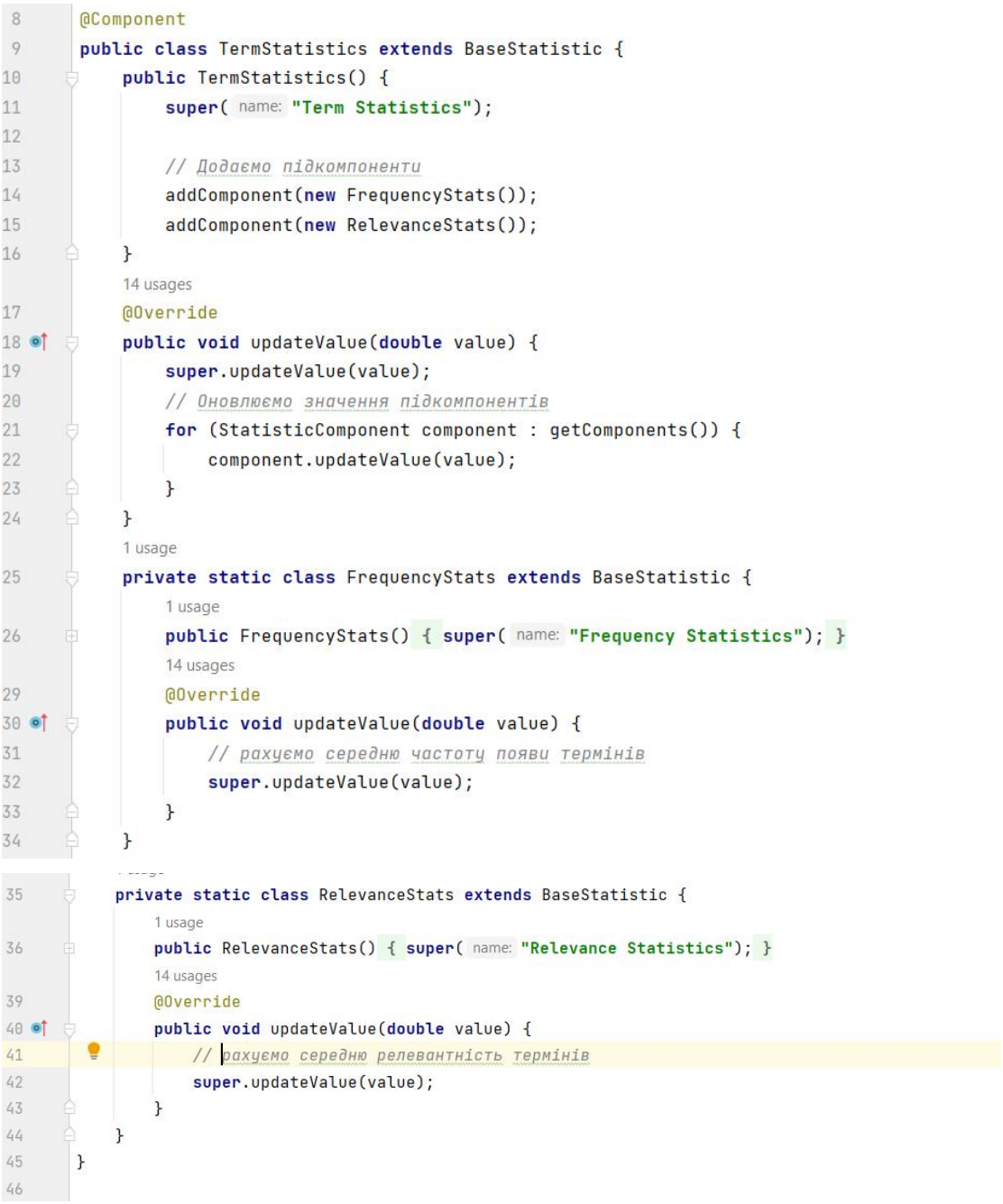
Рисунок 2 - Код класу BaseStatistics

Клас BaseStatistic є базовим класом для всіх компонентів статистики у цьому проекті. Реалізує інтерфейс StatisticComponent

```java
@Component
public class PageStatistics extends BaseStatistic {

    public PageStatistics() {
        super( name: "Page Statistics");

        // Додаємо підкомпоненти
        addComponent(new ContentStats());
        addComponent(new LinkStats());
        addComponent(new SemanticStats());
    }
    14 usages
    @Override
    public void updateValue(double value) {
        super.updateValue(value);
        // Оновлюємо значення підкомпонентів
        for (StatisticComponent component : getComponents()) {
            component.updateValue(value);
        }
    }
    1 usage
    private static class ContentStats extends BaseStatistic {
        1 usage
        public ContentStats() { super( name: "Content Statistics"); }
        14 usages
        @Override
        public void updateValue(double value) {
            // рахуємо середній розмір контенту
            super.updateValue(value);
        }
    }

     private static class LinkStats extends BaseStatistic {
         1 usage
         public LinkStats() { super( name: "Link Statistics"); }


         14 usages
         @Override
         public void updateValue(double value) {
             //рахуємо кількість посилань
             super.updateValue(value);
         }
     }
     1 usage
     private static class SemanticStats extends BaseStatistic {
         1 usage
         public SemanticStats() { super( name: "Semantic Page Statistics"); }


         14 usages
         @Override
         public void updateValue(double value) {
             // оновлення загальної кількості семантичних сторінок
             super.updateValue(super.getValue() + value); // Додаємо нове значення
         }
     }
}
```

Рисунок 3 - Код класу PageStatistics

Класу PageStatistics це композитний компонент для статистики сторінок. Ініціалізує підкомпоненти, такі як *ContentStats*, *LinkStats*, і *SemanticStats*. Оновлює значення для всіх підкомпонентів, що дозволяє централізовано керувати статистикою сторінок.



```
@Component
public class TermStatistics extends BaseStatistic {
    public TermStatistics() {
        super( name: "Term Statistics");

        // Додаємо підкомпоненти
        addComponent(new FrequencyStats());
        addComponent(new RelevanceStats());
    }
    14 usages
    @Override
    public void updateValue(double value) {
        super.updateValue(value);
        // Оновлюємо значення підкомпонентів
        for (StatisticComponent component : getComponents()) {
            component.updateValue(value);
        }
    }
    1 usage
    private static class FrequencyStats extends BaseStatistic {
        1 usage
        public FrequencyStats() { super( name: "Frequency Statistics"); }
        14 usages
        @Override
        public void updateValue(double value) {
            // рахуємо середню частоту появи термінів
            super.updateValue(value);
        }
    }
    private static class RelevanceStats extends BaseStatistic {
        1 usage
        public RelevanceStats() { super( name: "Relevance Statistics"); }
        14 usages
        @Override
        public void updateValue(double value) {
            // рахуємо середню релевантність термінів
            super.updateValue(value);
        }
    }
}
```

Рисунок 4 - Код класу PageStatistic.

Це композитний компонент для статистики термінів.

Ініціалізує підкомпоненти, такі як *FrequencyStats і RelevanceStats*.

Оновлює значення для всіх підкомпонентів, що дозволяє централізовано керувати статистикою термінів.

```java
@Component
public class CrawlerStatistics extends BaseStatistic {

    @Autowired
    public CrawlerStatistics(PageStatistics pageStats, TermStatistics termStats) {
        super( name: "Crawler Statistics");

        // Додаємо основні компоненти статистики
        addComponent(pageStats);
        addComponent(termStats);
    }
    1 usage
    public void updatePageStats(int pageSize, int linkCount) {
        for (StatisticComponent component : getComponents()) {
            if (component instanceof PageStatistics) {
                // Оновлюємо загальну статистику сторінок
                component.updateValue(pageSize);
            }
        }
    }
    1 usage
    public void updateTermStats(int termCount, double averageRelevance) {
        for (StatisticComponent component : getComponents()) {
            if (component instanceof TermStatistics) {
                // Оновлюємо загальну статистику термінів
                component.updateValue(termCount);
            }
        }
    }
    public void updateSemanticStats(int semanticPageCount) {
        for (StatisticComponent component : getComponents()) {
            if (component instanceof PageStatistics) {
                for (StatisticComponent subComponent : component.getComponents()) {
                    if (subComponent.getName().equals("Semantic Page Statistics")) {
                        subComponent.updateValue(semanticPageCount);
                        break;
                    }
                }
            }
        }
    }
    public double getOverallSemanticPageStats() {
        for (StatisticComponent component : getComponents()) {
            if (component instanceof PageStatistics) {
                for (StatisticComponent subComponent : component.getComponents()) {
                    if (subComponent.getName().equals("Semantic Page Statistics")) {
                        return subComponent.getValue();
                    }
                }
            }
        }
        return 0.0;
    }
    2 usages
    public double getOverallPageStats() {
        for (StatisticComponent component : getComponents()) {
            if (component instanceof PageStatistics) {
                return component.getValue();
            }
        }
    }
```

Рисунок 5 - Код класу CrawlerStatistics.

```
71      public double getOverallTermStats() {
72          for (StatisticComponent component : getComponents()) {
73              if (component instanceof TermStatistics) {
74                  return component.getValue();
75              }
76          }
77          return 0.0;
78      }
        1 usage
79      public void clearStatistics() {
80          for (StatisticComponent component : getComponents()) {
81              component.updateValue(0);
82              for (StatisticComponent subComponent : component.getComponents()) {
83                  subComponent.updateValue(0);
84              }
85          }
86      }
87      public Map<String, Double> getStatistics() {
88          Map<String, Double> stats = new HashMap<>();
89          for (StatisticComponent component : getComponents()) {
90              if (component instanceof PageStatistics) {
91                  stats.put("averagePageSize", component.getValue());
92              } else if (component instanceof TermStatistics) {
93                  stats.put("averageTermsPerPage", component.getValue());
94              }
95          }
96          return stats;
97      }
98  }
```

Рисунок 5.1 - Продовження коду класу CrawlerStatistics.

Це головний композитний компонент, який об'єднує всю статистику краулера. Ініціалізує основні компоненти статистики, такі як PageStatistics і TermStatistics. Оновлює статистику для нових сторінок і термінів, делегуючи ці оновлення відповідним підкомпонентам.

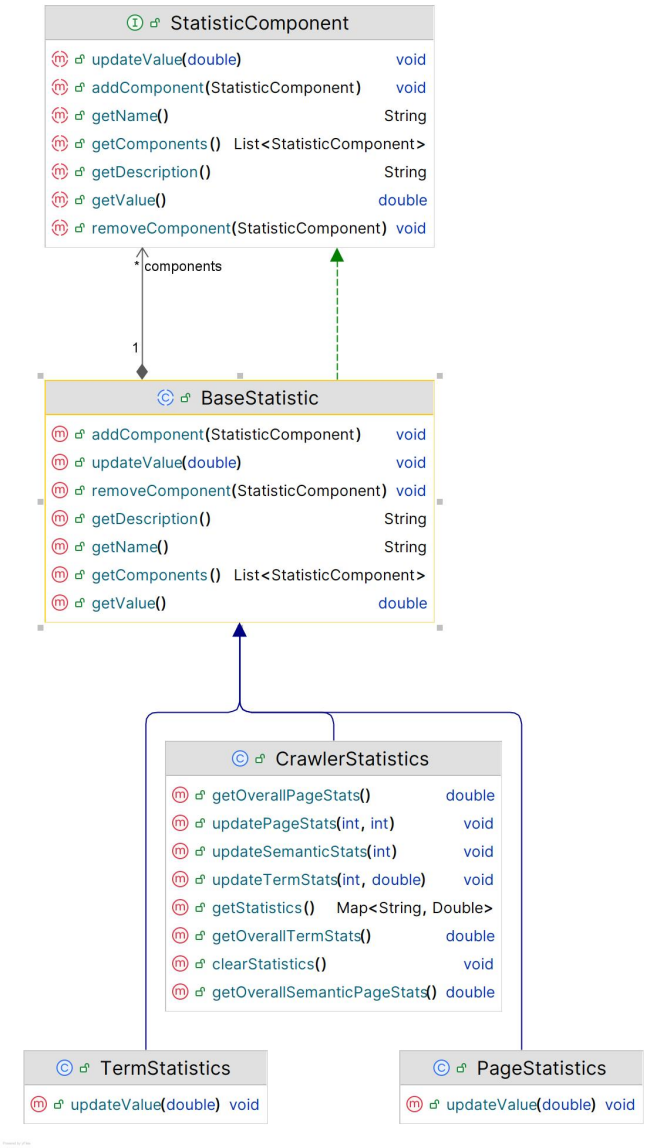


Рисунок 6 - Діаграма структури шаблону Composite

Діаграма демонструє шаблон Composite, дозволяє організувати статистику як деревоподібну структуру, де є основні компоненти та їх підкомпоненти.

Посилання на репозиторій:

https://github.com/mashunchik/Webcrawler

Висновок: У результаті лабораторної роботи було продемонстровано реалізацію шаблону Composite. Використання шаблону "Composite" у моєму проекті забезпечує ефективне управління статистичними даними, організуючи їх у вигляді ієрархії компонентів.

Шаблон дозволяє легко додавати нові компоненти або модифікувати існуючі, не змінюючи загальну структуру. Це сприяє підтримці та розширенню системи, що є критично важливим для довгострокових проектів.

Використання єдиного інтерфейсу для всіх компонентів статистики спрощує взаємодію з ними, дозволяючи обробляти окремі елементи та їх композиції однаково. Це знижує складність коду і робить його більш зрозумілим. Шаблон дозволяє централізовано керувати різними аспектами статистики, такими як сторінки, терміни і їх підкомпоненти. Це забезпечує гнучкість у зборі та обробці даних, що особливо важливо для динамічних систем, таких як веб-сканери.