



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**

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Патерни проектування»

Тема роботи: 7. Редактор зображень

Виконав

студент групи IA-33

Марченко Вадим Олександрович

Київ 2025

**Тема:** Патерни проектування

**Мета:** Вивчити структуру шаблонів «Composite», «Flyweight» (Пристосуванець), «Interpreter», «Visitor» та навчитися застосовувати їх в реалізації програмної системи

Посилання на репозиторій: <https://github.com/nwu1015/ImageEditor>

## Короткі теоретичні відомості

### Компонувальник (Composite)

Шаблон "Компонувальник" дозволяє працювати з групами об'єктів так само, як з одним об'єктом, організовуючи їх у деревоподібну структуру. Використовується, коли потрібно представити ієрархію частин-ціле і надати єдиний інтерфейс для роботи з окремими об'єктами та їх групами. У Java цей шаблон реалізується шляхом створення базового інтерфейсу або абстрактного класу для всіх об'єктів у структурі та підкласів для представлення як "листків" (окремих об'єктів), так і "гілок" (груп об'єктів). Це дозволяє клієнтському коду працювати з усією структурою без перевірок її деталей.

### Легковаговик (Flyweight)

Шаблон "Легковаговик" оптимізує використання пам'яті шляхом спільного використання об'єктів, які поділяють одинаковий стан. Використовується, коли система створює велику кількість однотипних об'єктів, що можуть розділяти загальний внутрішній стан. У Java цей шаблон зазвичай реалізується через фабрику, яка управлює пулом спільних об'єктів. Зовнішній стан об'єктів зберігається поза ними, щоб уникнути надмірного споживання пам'яті. Це дозволяє значно знизити накладні витрати в системах із великою кількістю об'єктів.

### Інтерпретатор (Interpreter)

Шаблон "Інтерпретатор" визначає спосіб представлення граматики ієрархії мови і надає інтерпретатор для її виконання. Використовується для роботи з мовами, які мають чітко визначену граматику, наприклад, для створення калькуляторів або розбору виразів. У Java цей шаблон реалізується через створення класів, які представляють правила граматики,

та методу `interpret()`, що виконує операції. Це забезпечує зручність у роботі з граматикою, проте може бути неефективним для складних мов через зростання кількості класів.

## **Відвідувач (Visitor)**

Шаблон "Відвідувач" дозволяє визначати нові операції для об'єктів без зміни їхніх класів. Використовується, коли необхідно виконати кілька різних операцій над об'єктами складної структури, але їх класи не можна змінювати. У Java цей шаблон реалізується шляхом створення інтерфейсу `Visitor`, який має методи для кожного типу елементів, та їх реалізацій для виконання конкретних операцій. Об'єкти структури реалізують метод `accept(Visitor visitor)`, який викликає відповідний метод відвідувача. Це дозволяє додавати нові операції, зберігаючи класи об'єктів незмінними.

## **Хід роботи**

### **7. Редактор зображень (state, prototype, memento, facade, composite, client-server)**

Редактор зображень має такі функціональні можливості: відкриття/збереження зображень у найпопулярніших форматах (5 на вибір студента), застосування ефектів, наприклад поворот, розтягування, стиснення, кадрування зображення, можливість створення колажів шляхом «нашарування» зображень.

У цій роботі я використовую патерн "Компонувальник" (Composite). Цей патерн було обрано як архітектурне рішення для того, щоб мати можливість поводитися з одиничними об'єктами (`ImageLayer`) та цілими групами об'єктів (`LayerGroup`) однаково.

У процесі роботи над проектом стало зрозуміло, що користувач може захотіти не лише маніпулювати окремими шарами, але й об'єднувати декілька шарів у групу, щоб застосовувати до них трансформації (такі як поворот, переміщення чи клонування) як до єдиного цілого.

Без використання патерну "Компонувальник", ця логіка привела б до значного ускладнення клієнтського коду, зокрема CollageService. Сервісу довелося б постійно перевіряти тип об'єкта за допомогою instanceof, щоб зрозуміти, чи має він справу з одним шаром чи з групою, що потребує рекурсивного обходу. Це створило б сильну зв'язаність CollageService з конкретними реалізаціями та зробило б код крихким і складним для підтримки.

Патерн "Компонувальник" вирішує цю проблему елегантно. Він надає єдиний спільний інтерфейс (у моєму проєкті – абстрактний клас LayerComponent) для всіх об'єктів у деревоподібній структурі.

Реалізація паттерну:

```
12  @Entity 2 inheritors  ± nwu1015
13  @Table(name = "layer_components")
14  @Inheritance(strategy = InheritanceType.SINGLE_TABLE)
15  @DiscriminatorColumn(name = "component_type")
16  @Data
17  public abstract class LayerComponent implements Prototype, Cloneable {
18      @Id
19      @GeneratedValue(strategy = GenerationType.IDENTITY)
20      private Long id;
21
22      @
23      private int positionX;
24      private int positionY;
25      private double rotationAngle = 0.0;
26      private int zIndex;
27
28      @ManyToOne(fetch = FetchType.LAZY)
29      @JoinColumn(name = "collage_id")
30      private Collage collage;
31
32      @
33      public abstract void render(Graphics2D g2d, ImageService imageService) throws IOException; 2 usages 1
34
35      @
36      public abstract void applyUpdate(CollageService.LayerUpdateDTO dto); 1 usage 2 implementations  ± nwu1015
37      @
38      public abstract ImageLayerMemento createMemento(); 3 usages 2 implementations  ± nwu1015
39
40      @
41      public abstract void restoreFromMemento(ImageLayerMemento memento); 2 usages 2 implementations  ± nwu1015
```

```

39         @Override 2 overrides  ± nwu1015
40 ⚡@  public LayerComponent clone() {
41     try {
42         return (LayerComponent) super.clone();
43     } catch (CloneNotSupportedException e) {
44         throw new RuntimeException("Can't clone component", e);
45     }
46 }
47 }
```

Рисунок 1. - Програмний код компоненту LayerComponent

```

16     @Entity  ± nwu1015
17     @DiscriminatorValue("LEAF")
18     @Data
19     @NoArgsConstructor
20     @EqualsAndHashCode(callSuper = true)
21     public class ImageLayer extends LayerComponent implements Prototype, Cloneable {
22         @ManyToOne(fetch = FetchType.LAZY)
23         @JoinColumn(name = "image_id")
24         private Image image;
25
26     @
27     private int width;
28     private int height;
29     private Integer cropX;
30     private Integer cropY;
31     private Integer cropWidth;
32     private Integer cropHeight;
33
34     @Column(columnDefinition = "TEXT")
35     private String effectsJson;
36
37     @Override 2 usages  ± nwu1015
38     public void render(Graphics2D g2d, ImageService imageService) throws IOException {
39         // Логіка малювання ОДНОГО шару
40         BufferedImage img = imageService.applyTransformationsToLayer(this.getId());
41         g2d.drawImage(img, getPositionX(), getPositionY(), observer: null);
42     }
```

Рисунок 2. - Частина програмного коду листка ImageLayer

```

15  @Entity  ± nwu1015
16  @DiscriminatorValue("COMPOSITE")
17  @Data
18  @NoArgsConstructor
19  @EqualsAndHashCode(callSuper = true)
20  public class LayerGroup extends LayerComponent {
21
22      @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER, orphanRemoval = true)
23      @JoinColumn(name = "parent_group_id")
24      @OrderBy("zIndex ASC")
25  ⚡  private List<LayerComponent> children = new ArrayList<>();
26
27
28  ⚡  public void render(Graphics2D g2d, ImageService imageService) throws IOException {
29      for (LayerComponent child : children) {
30          child.render(g2d, imageService);
31      }
32  }
33
34
35  ⚡  public void applyUpdate(CollageService.LayerUpdatedDTO dto) {
36      for (LayerComponent child : children) {
37          child.applyUpdate(dto);
38      }
39  }
40
41
42  ⚡  public LayerComponent clone() {
43      LayerGroup newGroup = (LayerGroup) super.clone();
44      newGroup.setId(null);
45
46      newGroup.setChildren(new ArrayList<>());
47      for (LayerComponent child : this.children) {
48          LayerComponent clonedChild = child.clone();
49          newGroup.add(clonedChild);
50      }
51      return newGroup;
52  }
53
54
55  ⚡  public ImageLayerMemento createMemento() {
56      throw new UnsupportedOperationException("Memento not supported for this groups");
57  }
58
59
60  ⚡  public void restoreFromMemento(ImageLayerMemento memento) {
61      throw new UnsupportedOperationException("Memento not supported for this groups");
62  }
63
64      public void add(LayerComponent component) {  ± nwu1015
65          children.add(component);
66      }
67  }

```

Рисунок 3. - Програмний код компонувальника LayerGroup

У моїй реалізації:

- LayerComponent (Компонент) визначає спільні операції, такі як render(), applyUpdate() та clone().
- ImageLayer (Листок) є класом, що успадковує LayerComponent і надає конкретну реалізацію для одного об'єкта-зображення.
- LayerGroup (Компонувальник) також успадковує LayerComponent, але містить у собі список дочірніх LayerComponent. Його реалізація методів, як-от render(), просто делегує виклик усім своїм "нащадкам".

Клас Collage (Клієнт) тепер містить List<LayerComponent>, не розрізняючи, чи є елемент списку "Листком" чи "Компонувальніком". Коли CollageService викликає component.render(), йому байдуже, чи це один ImageLayer, чи LayerGroup, що містить 50 інших об'єктів.

### Діаграма класів

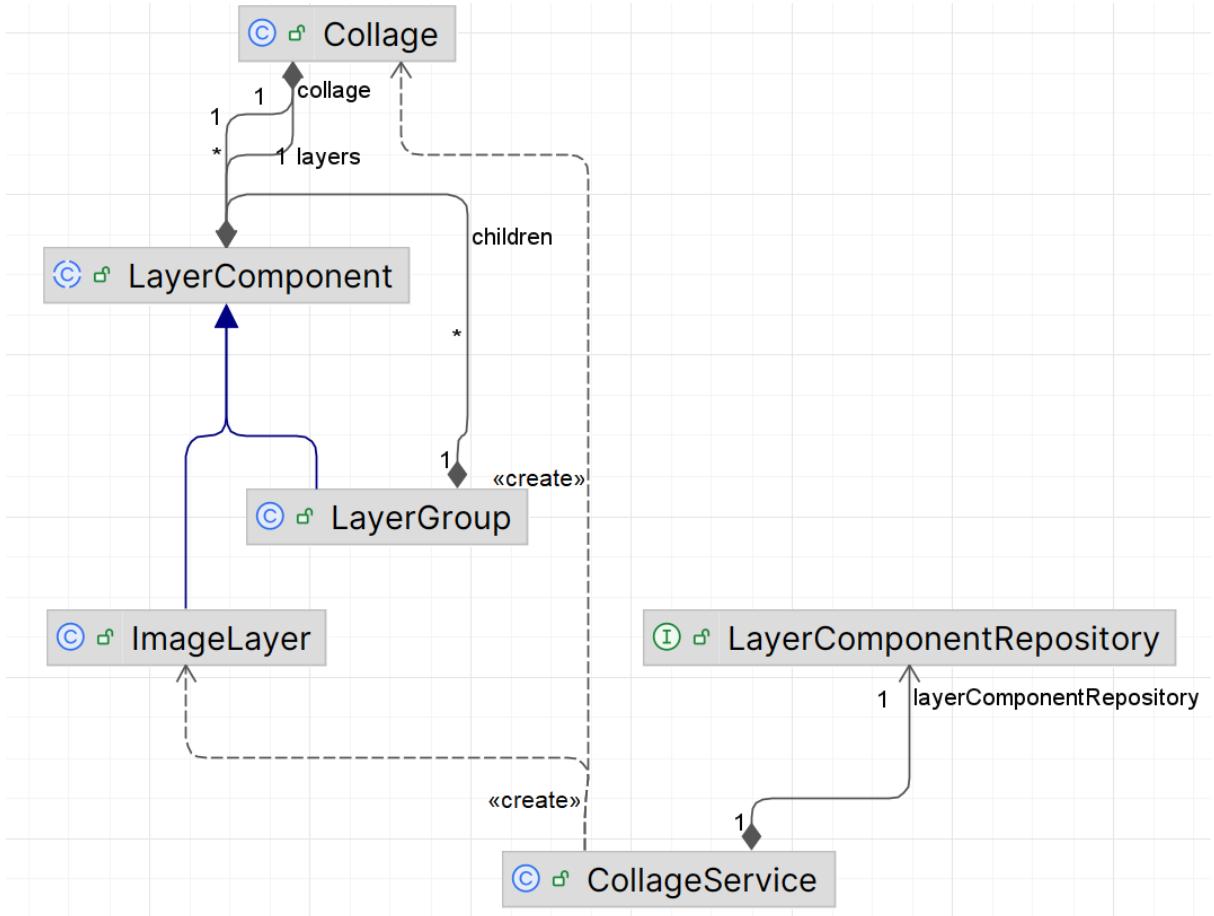


Рисунок 4. - Діаграма класів при реалізації паттерну проєктування «Composite»

**Висновок:** Виконуючи цю лабораторну роботу, я ознайомився з такими патернами, як «Composite», «Flyweight», «Interpreter» та «Visitor».

Особливу увагу я приділив реалізації шаблону "Компонувальник" (Composite), детально описавши його основну логіку та функціональність у контексті моєго проєкту. Цей патерн було обрано як архітектурне рішення для того, щоб мати можливість однаково поводитися з одиничними об'єктами (шарами ImageLayer) та цілими групами об'єктів (новий клас LayerGroup).

Під час реалізації шаблону "Компонувальник" я зрозумів, наскільки цей патерн елегантно вирішує проблему складної, деревоподібної структури об'єктів. Він дозволяє "Клієнту" (CollageService та Collage) працювати

через єдиний абстрактний інтерфейс (LayerComponent), не розрізняючи, чи є об'єкт одиничним "Листком" (ImageLayer), чи "Компонувальником" (LayerGroup), що містить інших "нащадків". Це робить код клієнта значно чистішим, оскільки зникає потреба у складних перевірках instanceof та рекурсивних обходах усередині сервісу.

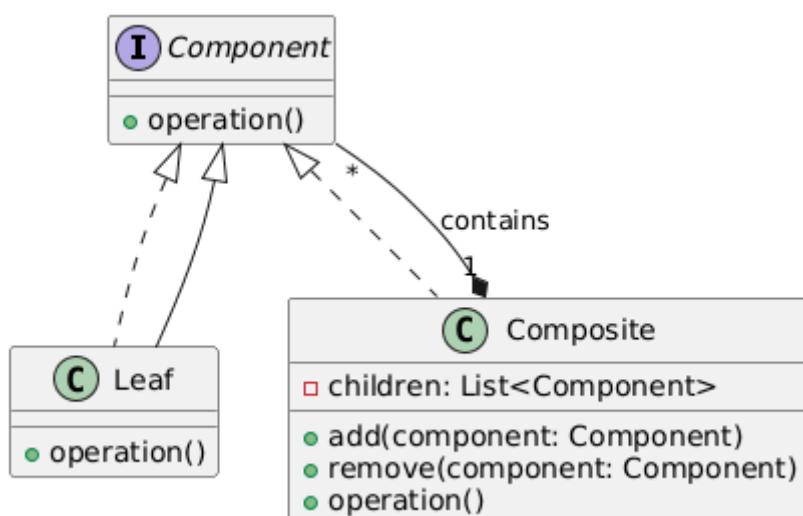
## Контрольні запитання

1. Яке призначення шаблону «Композит»?

Шаблон «Композит» (Composite) призначений для того, щоб єдиним способом працювати як з окремими об'єктами, так і з групами об'єктів.

Він дозволяє будувати деревоподібні структури (наприклад, елементи інтерфейсу, файлову систему, шари зображення) і викликати операції однаково для листків та компонувальників. Тобто клієнт не думає, має він справу з одним елементом чи з цілим набором елементів — інтерфейс у них спільний.

2. Нарисуйте структуру шаблону «Композит».



3. Які класи входять в шаблон «Композит», та яка між ними взаємодія?

Шаблон «Композит» складається з трьох основних класів: Компонент, Листок і Компоновальник.

Компонент – це базовий інтерфейс або абстрактний клас, який визначає спільні операції для всіх елементів структури.

Листок – це простий елемент без дочірніх об'єктів, який просто виконує свою частину роботи.

Компоновальник — це елемент, який може містити інші компоненти. Він зберігає список дітей і викликає їхні операції, коли виконується його власна операція.

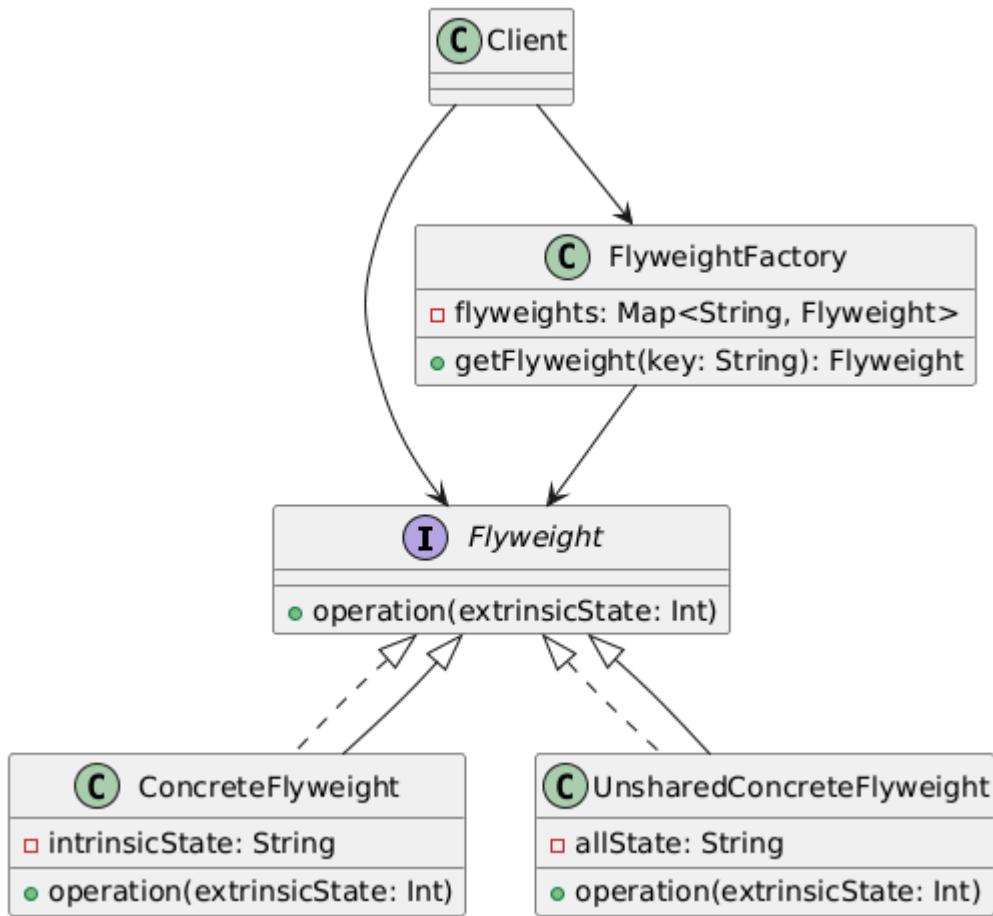
Взаємодія між ними така: клієнт звертається до всіх елементів через тип «Компонент». Листки обробляють запити напряму, а компоновальники передають їх своїм дочірнім елементам. Завдяки цьому клієнт не відрізняє, працює він з одним елементом чи з цілою групою.

#### 4. Яке призначення шаблону «Легковаговик»?

Шаблон «Легковаговик» (Flyweight) призначений для зменшення використання пам'яті, коли у програмі потрібно створити дуже багато однотипних об'єктів.

Замість того щоб створювати тисячі одинакових об'єктів, спільні дані виносять у один розділюваний об'єкт, а унікальні дані зберігають окремо. Це дозволяє значно економити ресурси та прискорювати роботу програми.

#### 5. Нарисуйте структуру шаблону «Легковаговик».



6. Які класи входять в шаблон «Легковаговик», та яка між ними взаємодія?

У шаблон «Легковаговик» входять такі основні класи:

- Flyweight (Легковаговик) – об'єкт, який містить *внутрішній стан* (той, що спільний і не змінюється). Ці об'єкти можна багаторазово використовувати.
- ConcreteFlyweight – конкретна реалізація легковаговика з фіксованими внутрішніми даними.
- FlyweightFactory – створює та зберігає легковаговики. Якщо об'єкт з такими даними уже є, фабрика повертає існуючий, замість створення нового.
- Client – використовує легковаговики, передаючи їм *зовнішній стан* (унікальні дані, які не можна зберігати всередині легковаговика).

Клієнт звертається до фабрики, фабрика повертає вже існуючий легковаговик або створює новий. Легковаговик містить лише спільні дані, а унікальні дані клієнт передає йому під час використання. Це дозволяє значно економити пам'ять.

## 7. Яке призначення шаблону «Інтерпретатор»?

Шаблон «Інтерпретатор» (Interpreter) призначений для опису граматики мови та інтерпретації виразів цієї мови.

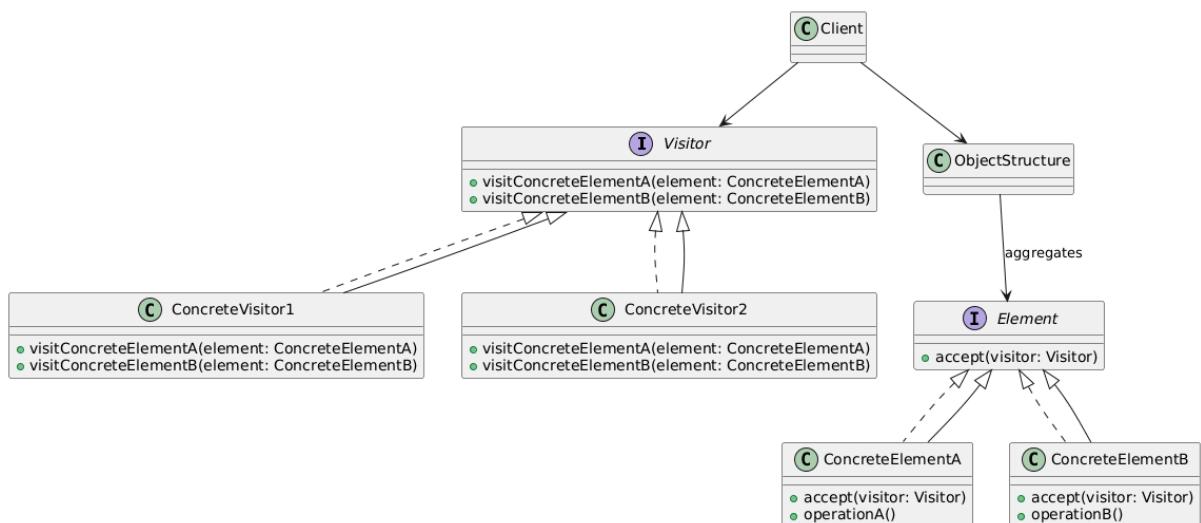
Він дозволяє створити невелику мову (наприклад, для формул, фільтрів, команд) та написати набір класів, які можуть читати та виконувати вирази цієї мови.

## 8. Яке призначення шаблону «Відвідувач»?

Шаблон «Відвідувач» (Visitor) призначений для того, щоб додавати нові операції до об'єктів складної структури, не змінюючи їхні класи.

Тобто він дозволяє винести логіку обробки об'єктів у окремий клас-відвідувач і “проходити” ним по елементах структури, виконуючи потрібні дії.

## 9. Нарисуйте структуру шаблону «Відвідувач».



10. Які класи входять в шаблон «Відвідувач», та яка між ними взаємодія?

Шаблон «Відвідувач» складається з таких основних класів:

- **Visitor** (Відвідувач)

Оголошує методи для відвідування кожного типу елементів у структурі.

Кожен метод відповідає конкретному класу елементів.

- **ConcreteVisitor**

Реалізує конкретні операції, які виконуються над елементами.

Наприклад: підрахунок, вивід, перевірка, експорт тощо.

- **Element** (Елемент)

Базовий інтерфейс або абстрактний клас для всіх об'єктів структури.

Містить метод accept(Visitor).

- **ConcreteElement**

Конкретні елементи, які реалізують метод accept, передаючи себе відвідувачу (visitor.visit(this)).

- **ObjectStructure**

Колекція або дерево елементів, які можна “обійти” за допомогою відвідувача.

Кожен елемент має метод accept, який приймає відвідувача. Відвідувач заходить у елемент, а елемент викликає відповідний метод відвідувача. Так відвідувач може виконувати нові операції над елементами, не змінюючи їхні класи.