Visitor

A behavioral pattern

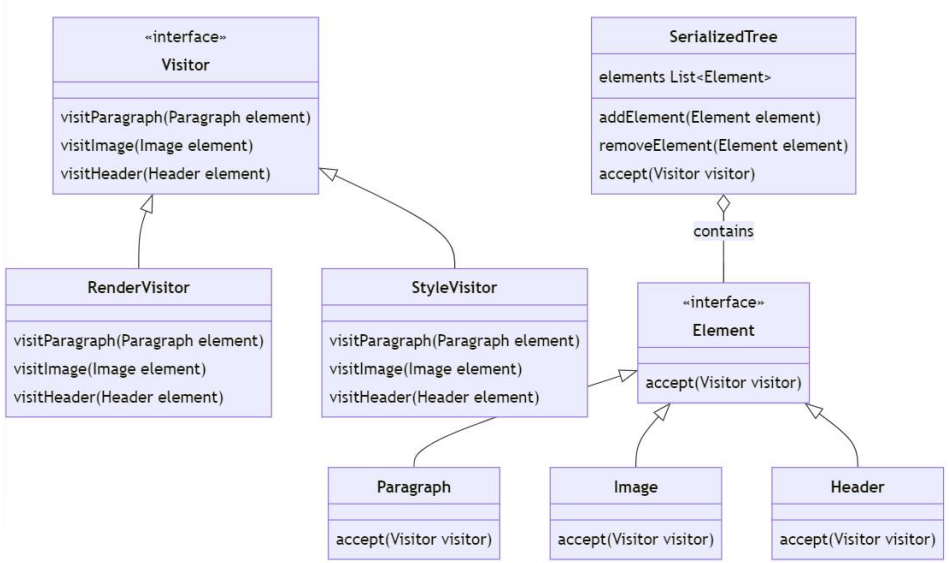
Learning goals

1. Learn the idea, structure, and Java implementation of the Visitor design pattern.
2. Learn to apply the Visitor DP in your own programming.

Idea of Visitor

- The Visitor design pattern allows adding new operations to existing object structures without modifying them.
- It separates an algorithm from the object structure it operates on
  - This promotes loose coupling.
- Particularly useful when dealing with complex object structure, like a composite object.

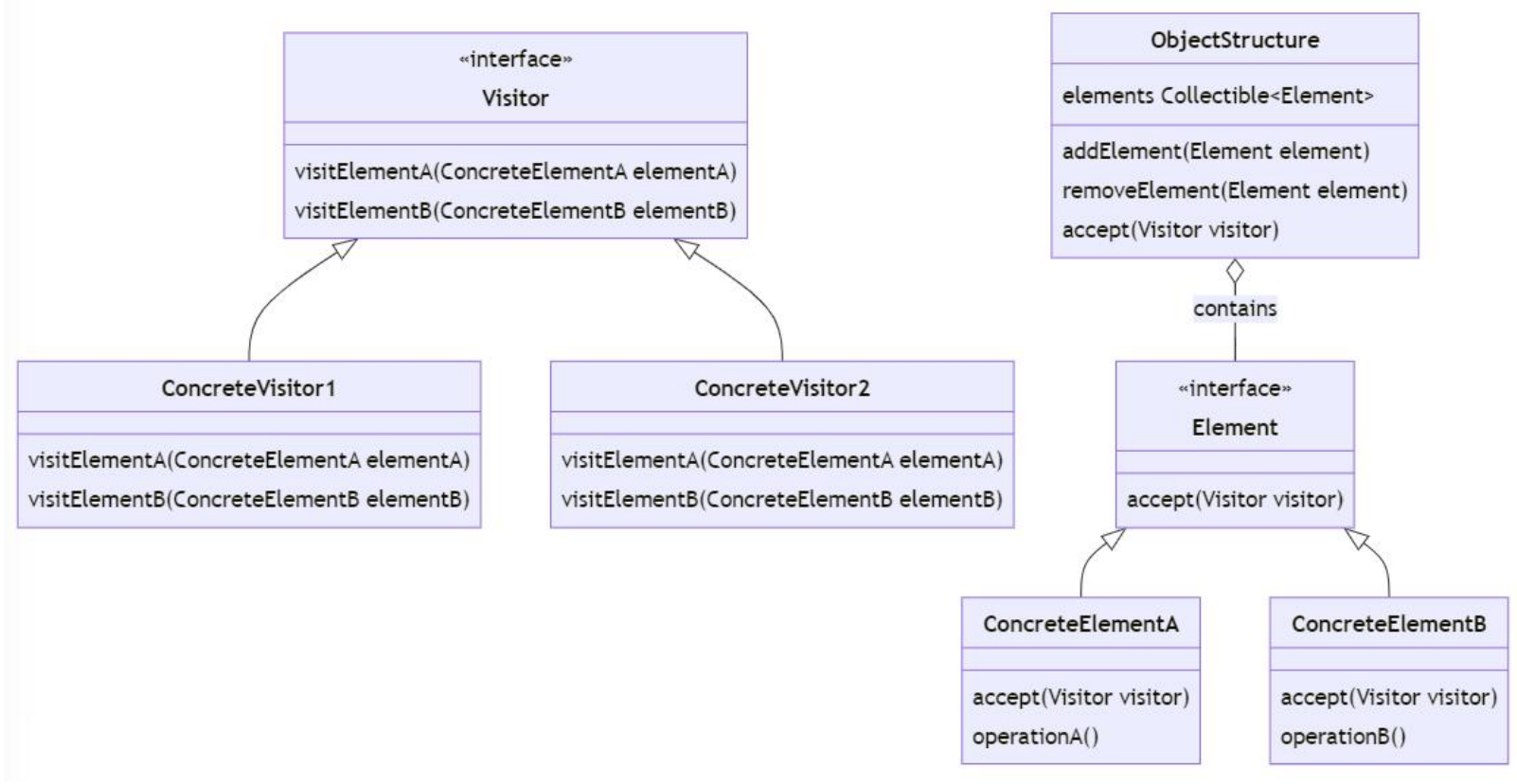
Example: HTML editor



- With an HTML editor, you can view individual elements in the DOM tree, rendered and styled.
  - There are Paragraph, Image and Header elements. Each element can be rendered and styled.
- The render() and style() methods are not hard-coded into the corresponding classes but implemented with Visitors.

Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 208

General structure



Roles

- **Visitor**: Declares a visit operation for each class of ConcreteElement in the object structure.
  - The operation's name and signature identify the class that sends the visit request to the visitor.
- **Concrete Visitor**: Implements each operation declared by Visitor. Each operation implements a fragment of the algorithm defined for the corresponding class of object in the structure.
- **Element**: Defines an accept operation that takes a visitor as an argument.
- **Concrete Element**: Implements an accept operation that takes a visitor as an argument.

Visitors and state information

- Visitors can accumulate state as they traverse and Object Structure.
- Example: calculate the sum of areas of the shapes.

```
public interface ShapeVisitor {
   void visit(Circle circle);
   void visit(Rectangle rectangle);
}
```

```
public class AreaAccumulator implements ShapeVisitor {
   private double totalArea = 0;

   public void visit(Circle circle) {
      totalArea += Math.PI * circle.getRadius() *
circle.getRadius();
   }

   public void visit(Rectangle rectangle) {
      totalArea += rectangle.getWidth() *
rectangle.getHeight();
   }

   public double getTotalArea() {
      return totalArea;
   }
}
```

Practical issues

- The Visitor pattern has two class hierarchies that don't have to be connected.
- Adding new functionality is easy.
  - Just add a new subclass to the Visitor interface.
- It is laborious to add a new Concrete Element.
  - A new method must be coded into each Visitor.
- May break encapsulation.
  - Often enforces creation of public getters for Concrete Elements so enable Visitors carry out their tasks.