VISITOR DESIGN PATTERN - SPACE JUNK (GROUP# 1)

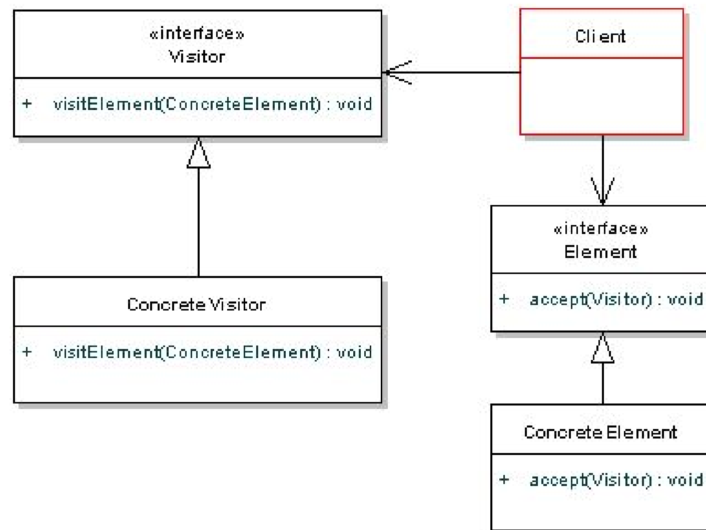
Ahmad Salar Gohar asgohar
Vidhyasagar Harihara v2mahade
Philip Young p5young

The visitor pattern is a behavioral design pattern, which means it affects how objects interact with each other. Behavioral design patterns manage the algorithms, relationships and responsibilities between objects.

The Gang of Four, who wrote the original book on design patterns, define the purpose of visitor pattern as "Allowing for one or more operation to be applied to a set of objects at runtime, decoupling the operations from the object structure." In less technical terms, the visitor pattern can be employed where an operation or a set of operations need to be applied to objects of a certain type. The objects affected are not changed themselves; the added operational ability is solely the responsibility of the visitor itself.

HOW VISITOR PATTERN WORKS

The following UML diagram shows the basic structure of the visitor pattern. It also defines some of the specific terminology that the pattern uses. The Client is the object that is requesting some action to be performed on a set of related objects, here referenced as Element. Element is an interface which is then implemented by many ConcreteElements, which are the objects on which a new operation is desired. This new operation can be introduced by a Visitor object. The Visitor itself is an interface which declares a visitElement(ConcreteElement) method for each ConcreteElement in the Element object hierarchy. The Element interface also defines an accept method which takes a Visitor object as parameter. The Visitor object's visitElement is then called and the object calling it passes itself as the argument so that the right method is called. The UML shows this relationship and the mentioned methods.



The intended use case for the Visitor pattern is to extend the set of operations that can be performed on a set of objects with minimal changes to the set of objects.
Visitor allows us to decouple the functionalities on the data from the data structures themselves. The pattern allows the design to respect cohesion as data structure classes are simpler (they have fewer methods) and the functionalities are encapsulated into Visitor implementations. This is done via double-dispatching: using accept() methods in the structure classes (ConcreteElement) and visit() methods in the ConcreteVisitor classes. Double-dispatch means that the function (visit) that will be called will be determined at runtime. This pattern enables future changes on the Element objects because a new operation on those objects can be defined without severe changes.

BENEFITS OF USING VISITOR

- Add functions to class libraries for which you either do not have the source or cannot change the source
- Obtain data from a disparate collection of unrelated classes and use it to present the results of a global calculation to the user program
- Gather related operations into a single class rather than changing or deriving classes to add these operations

- It is simpler to substitute a set of operations by substituting the ConcreteVisitor that visits the Elements.

DRAWBACKS OF USING VISITOR

- In case of an unwieldy number of ConcreteElements, the number of visit methods for each ConcreteElement can become too large and this will affect complexity
- For a smaller number of ConcreteElements (< 4), it might be overkill to introduce a separate class, especially if the added operation is simple enough to be added to the class itself
- The arguments and return types for the visiting methods needs to be known in advance, so the Visitor pattern is not good for situations where these visited classes are subject to change. Every time a new type of Element is added, every Visitor derived class must be amended.
- It can be difficult to refactor the Visitor pattern into code that wasn't already designed with the pattern in mind. For example, the object hierarchy of the Element class may not be as clean as the UML diagram and complexities could be introduced.

NFPs IMPROVED

- Complexity
- Evolvability / Extensibility

NFPs DEGRADED

- Efficiency
- Reusability

EXAMPLE:

We will be presenting 3 different scenarios to explain the Visitor pattern.

<u>Scenario 1: The Visit of the Empire</u>

The Empire sends a convoy (Visitor) to visit planets (ConcreteElements) of interest and act on them accordingly. This example servers as a parallel to the Visitor pattern because the convoy has a different way of interacting with each of the planets.

Scenario 2: Boarding Pass

The same convoy is now a guard at the airport. This example serves to show that the same convoy is not able to handle tasks on a new set of Elements. It has been tailored to particular set of objects, and new Visitor would be required for a different set of objects.

Scenario 3: Security Consultant

A company hires a security consultant who does a poor job but replacing the consultant identifies the security issues in the company. In this example the Visitor is the consultant and the company is the Element object hierarchy. This shows that if different (in this case, also better) operations are requested, they can be implemented in a different Visitor.