Software Engineering Design Patterns (2): Visitor Pattern

Erik Fredericks // frederer@gvsu.edu

Adapted from materials provided by Byron DeVries, Jagadeesh Nandigam

or, what happens if we need to add things to a class that we can't change?

Visitor pattern

Pattern Category: Behavioral

Intent: Represent an operation to be performed on the elements of an object structure.

Problem addressed:

- Data structures can have many operations that could be performed, but aren’t built into the data structure.

Solution:

- Make operations separate objects and handle a known data structure.

Visitor pattern

Pattern Category: Behavioral

Implementation:

- Each data structure has a visitor class, each element type has an operation defined within the visitor.

Consequences:

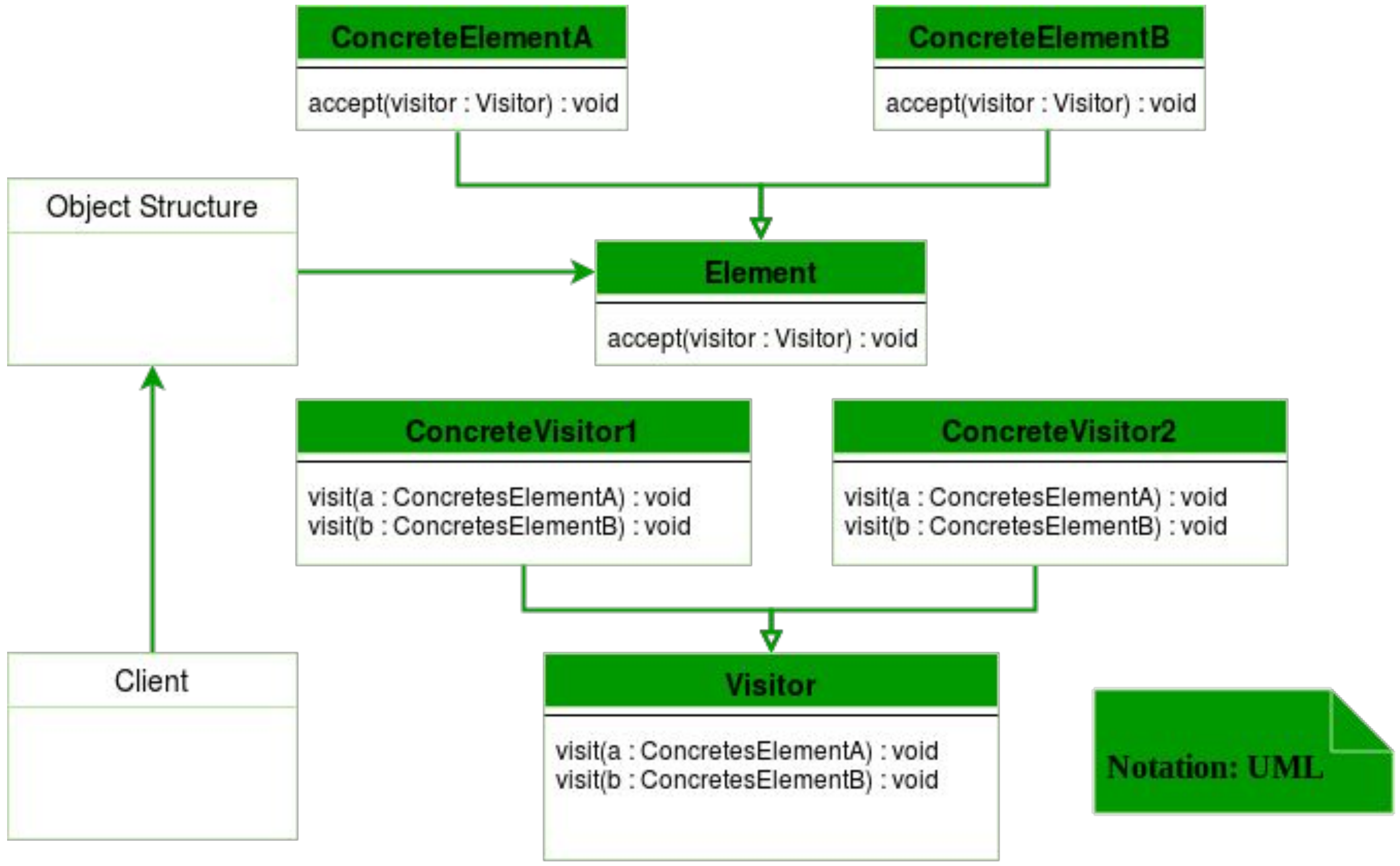
- Visitor makes adding new operations easy
- Visitor gathers related operations and separates unrelated ones
- Adding new elements to the data structure is hard (impact across visitors)
- Visitors accumulate state (akin to global variables)
- Visitors break encapsulation by performing an operation an element would normally do itself

Why visitor?

Class is fixed and cannot be changed

Separate algorithm from object structure

Visitor Pattern: UML (Implementation)


ConcreteElementA
accept(visitor : Visitor) : void
ConcreteElementB
accept(visitor : Visitor) : void
Object Structure
Element
accept(visitor : Visitor) : void
ConcreteVisitor1
visit(a : ConcretesElementA) : void
visit(b : ConcretesElementB) : void
ConcreteVisitor2
visit(a : ConcretesElementA) : void
visit(b : ConcretesElementB) : void
Client
Visitor
visit(a : ConcretesElementA) : void
visit(b : ConcretesElementB) : void
Notation: UML

https://www.geeksforgeeks.org/visitor-method-python-design-patterns/

Client:

- Consumer of the classes of the visitor design pattern.
- It can access the data structure objects and can instruct them to accept a visitor for the future processing.

Visitor:

- An Abstract class which is used to declare visit operations for all visitable classes.

Concrete Visitor:

- Each Visitor will be responsible for different operations. For each type of visitor all the visit methods, declared in abstract visitor, must be implemented.

(Visitable == Element (from Figure))

Visitable:

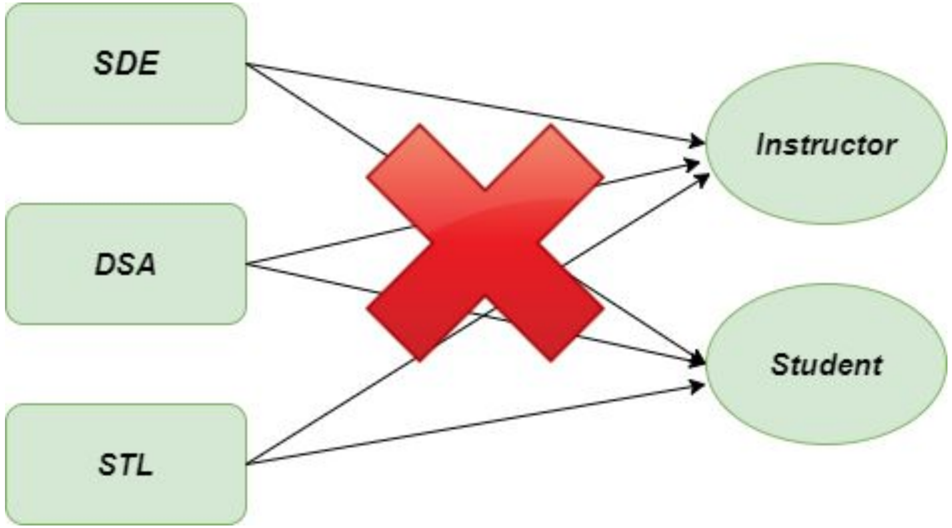
- `accept` operations is declared by this class.
- Acts as the entry point which enables an object to be visited by visitor.

Concrete Visitable:

- Implement the Visitable class and defines the accept operation.
- The visitor object is passed to this object using the `accept` operation.

Sample

Tying instructors and students to courses and figuring out who is responsible for what



https://www.geeksforgeeks.org/visitor-method-python-design-patterns/

Advantages

Open/Closed principle:

- New behaviors can be easily added to different classes without changing those classes

Single Responsibility principle:

- Multiple behaviors can be added to same class

Addition of entities:

- Update visitor without impacting base class

Updating logic:

- If operational logic is updated, only change visitor rather than all classes

Still https://www.geeksforgeeks.org/visitor-method-python-design-patterns/

Disadvantages

Lots of updates:

- Update each visitor when a class is added/removed

Hard to extend:

- Too many visitors makes it hard to extend class interface

Lack of access:

- Data hiding principles may prohibit visitors from accessing fields

Uses?

Recursive structures:

- Works nicely with recursion (e.g., XML, directory trees, JSON, etc.)
- Visit each node recursively

Performing operations:

- Call visitor method when performing operations on tree nodes, linked list nodes, etc.

Demo time!

Java code: https://www.baeldung.com/java-visitor-pattern

Pretend like you're making a phone app

It is a world-changing social media application that allows you to manage all of your multiplayer game characters (cross-game no less) in one simple interface. You load the application, it pulls data from each multiplayer game and shows your avatar, complete with inventory and stats. Here, you can minmax your character stats to your heart's content.

~~Join the CIS350 Discord discussion channels~~ Group up and come up with (describe, don't make a diagram):

- One **use case** for an **Observer** pattern
- One **use case** for a **Visitor** pattern

Write everybody's name down and submit at the end of class (or, the day) in Blackboard