DWARVES
FOUNDATION

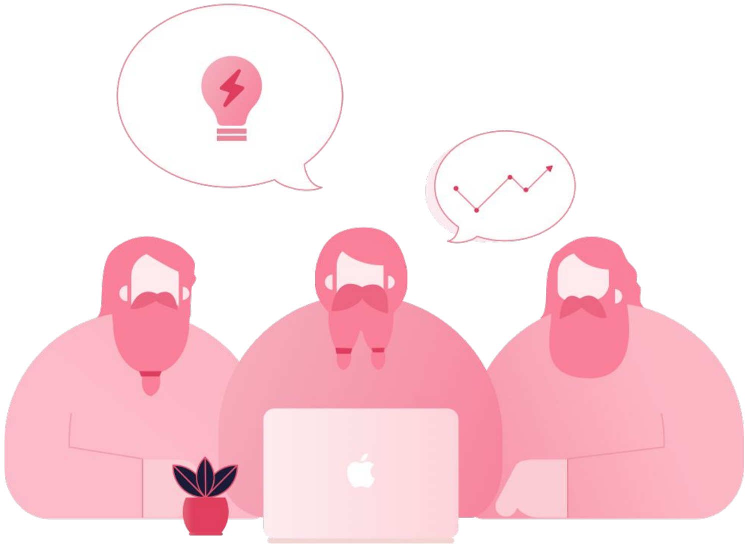
Visitor design pattern

Behavioral design pattern that lets you separate algorithms from the objects on which they operate.

Tai Pham

Software Engineer

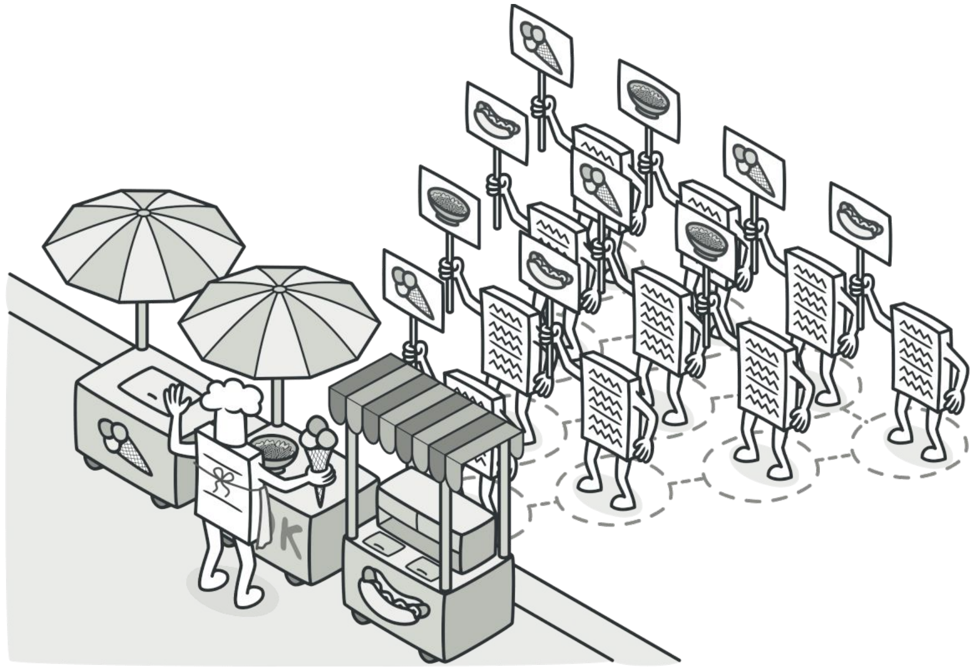
taipn@d.foundation



Agenda

1. Problem - Solution
2. What is the visitor design pattern
3. Applicability
4. Pros and Cons
5. Q&A

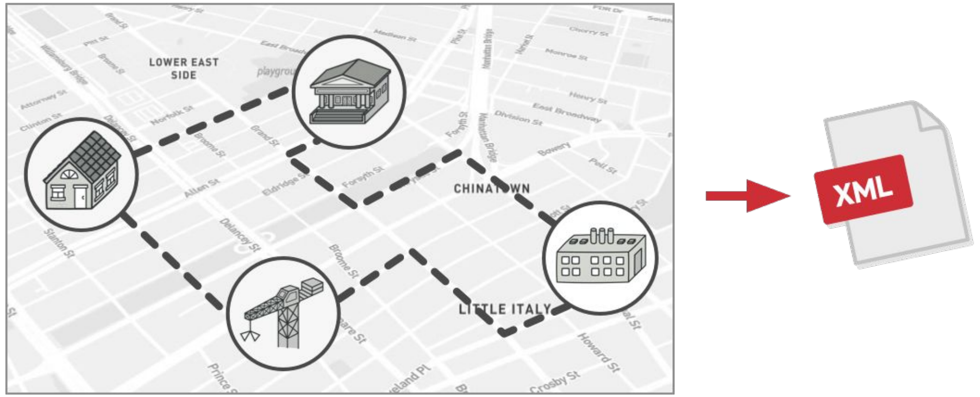
Problem



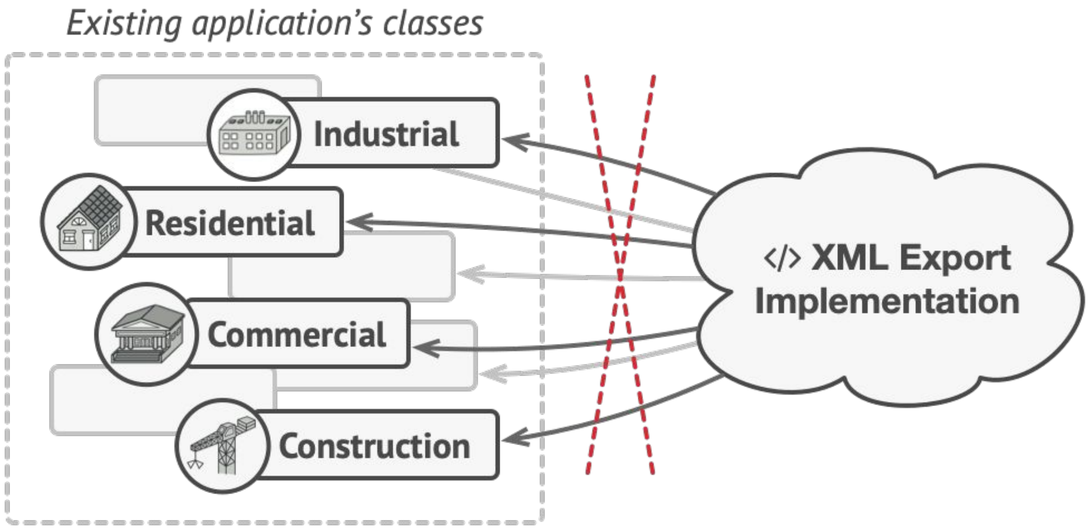
DWARVES FOUNDATION

Geographic information?

- Export nodes to?
- XML
- PDF
- JSON



Geographic information?



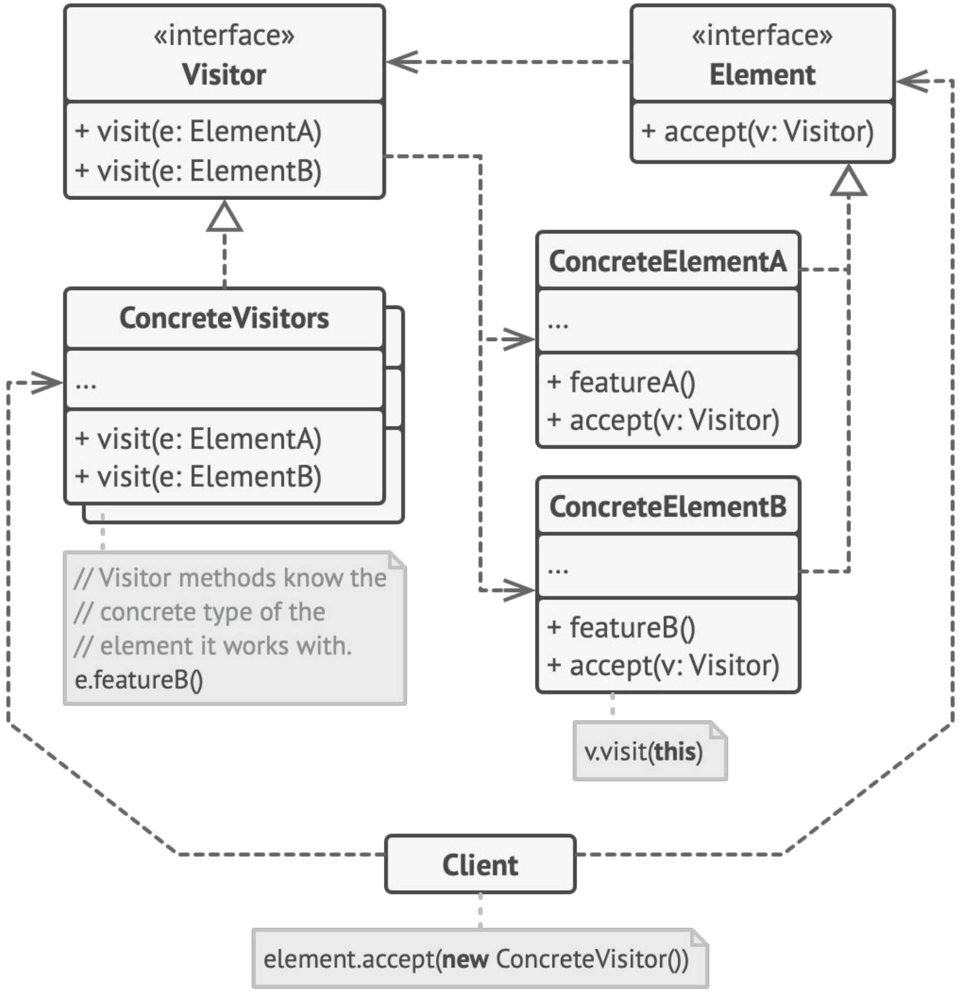
Solution- Visitor design pattern

- The Visitor pattern suggests that you place the new behavior into a separate class called visitor, instead of trying to integrate it into existing classes. The original object that had to perform the behavior is now passed to one of the visitor’s methods as an argument, providing the method access to all necessary data contained within the object.

Structure

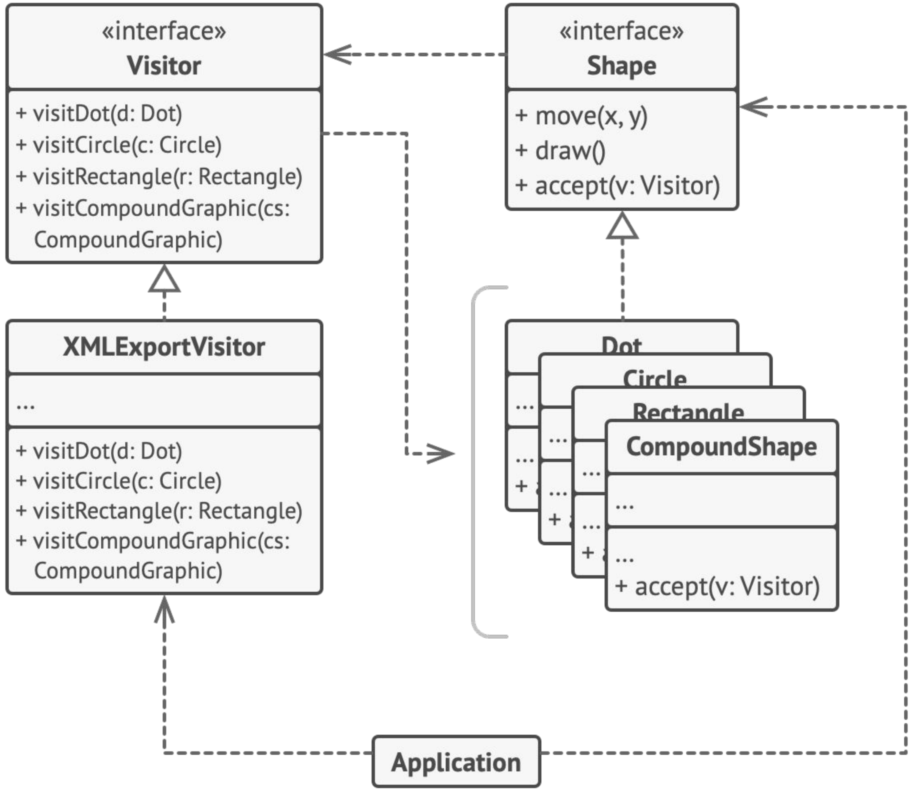
The Visitor interface declares a set of visiting methods that can take concrete elements of an object structure as arguments. These methods may have the same names if the program is written in a language that supports overloading, but the type of their parameters must be different.

Each Concrete Visitor implements several versions of the same behaviors, tailored for different concrete element classes.



DWARVES FOUNDATION

Pseudocode



Exporting various types of objects into XML format via a visitor object.

DWARVES FOUNDATION

Applicability

- **Use the Visitor when you need to perform an operation on all elements of a complex object structure (for example, an object tree).**
  The Visitor pattern lets you execute an operation over a set of objects with different classes by having a visitor object implement several variants of the same operation, which correspond to all target classes.

- **Use the Visitor to clean up the business logic of auxiliary behaviors.**
  The pattern lets you make the primary classes of your app more focused on their main jobs by extracting all other behaviors into a set of visitor classes.

- **Use the pattern when a behavior makes sense only in some classes of a class hierarchy, but not in others.**
  You can extract this behavior into a separate visitor class and implement only those visiting methods that accept objects of relevant classes, leaving the rest empty.

Pros and Cons

- *Open/Closed Principle*. "Open for extension" / "Closed for modification" You can introduce a new behavior that can work with objects of different classes without changing these classes.
- *Single Responsibility Principle*. "A module should be responsible to one, and only one, actor." You can move multiple versions of the same behavior into the same class.

Disadvantages

- You need to update all visitors each time a class gets added to or removed from the element hierarchy.
- Visitors might lack the necessary access to the private fields and methods of the elements that they’re supposed to work with

DWARVES FOUNDATION

Reference

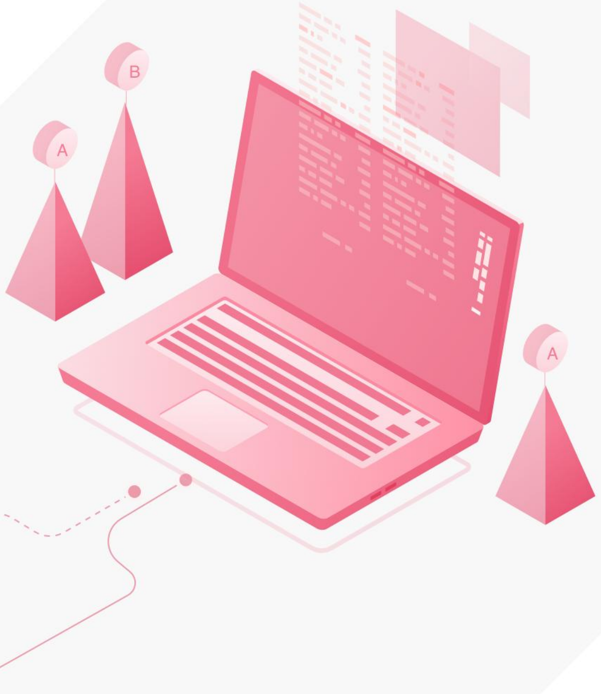
Resources & Reference links

- https://refactoring.guru/design-patterns/visitor
- https://refactoring.guru/design-patterns/visitor/typescript/example
-

DWARVES FOUNDATION

DWARVES
FOUNDATION

Thank You



DWARVES FOUNDATION

Q&A