

# Advanced Object-Oriented Design

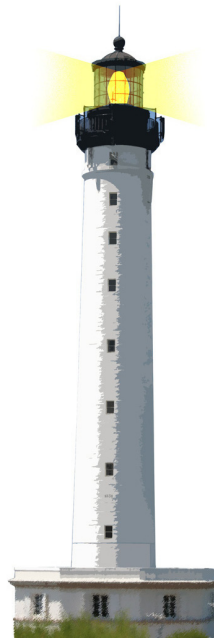
## Visitor

Modular and extensible first class actions

S. Ducasse



<http://www.pharo.org>



# Goal

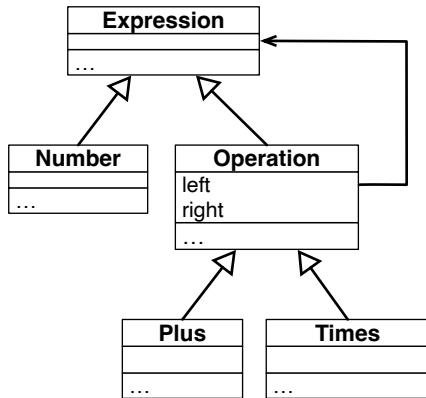
- Studying examples
- Understanding Visitor
- Example
- Pros and Cons



# Example: basic arithmetic expressions

Imagine a simple mathematical system

- a Composite
- with number and operation expressions



# Some expressions

1

ENumber value: 1

$(3 * 2)$

Times left: (ENumber value: 3) right: (ENumber value: 2)

$1 + (3 * 2)$

Plus

left: (ENumber value: 1)

right: (Times left: (ENumber value: 3) right: (ENumber value: 2))

In Pharo we can just extend Number so no need of ENumber value: but this is a detail



# Operations on the structure

We want to evaluate expressions, and print them

Evaluating

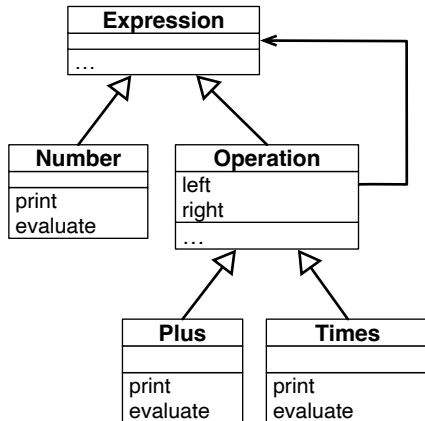
$1 + (3 * 2)$   
gives 7

Printing

$+1*32$



# First design: behavior define in the domain



# First design analysis

What if we need a stack to print well the expressions?

- Should we define the stack **in** the expression classes even if this is related **only** to print?

Why should we **mix** the information about the treatment of items and items themselves?

- What if we need a table for mathematical expressions **specific** to the LaTeX generation?
- What if we need a table for mathematical expression **specific** to the RDF generation?



# Let us see on a real system: Pillar

We have

- the core hierarchy is about 50 classes
- export to LaTeX (two versions)
- export to HTML
- export to Beamer
- export to ASCIIDoc, Markdown, Microdown
- transform trees for expansion





# First design conclusion

Putting all the behavior inside the domain objects

- **Blows up** the class API / state / methods
- **Mixes** concerns
- Is **not modular**: we cannot have one operation only
- **Prevents extension**: To add a new behavior I should change the domain



# Alternate design: using a visitor

## A Visitor:

- decouples operation from the structure
- represents an operation

It:

- supports modularity
  - can package visitors in separate packages
- supports extension
  - defines an extension protocol (set of messages to be defined)

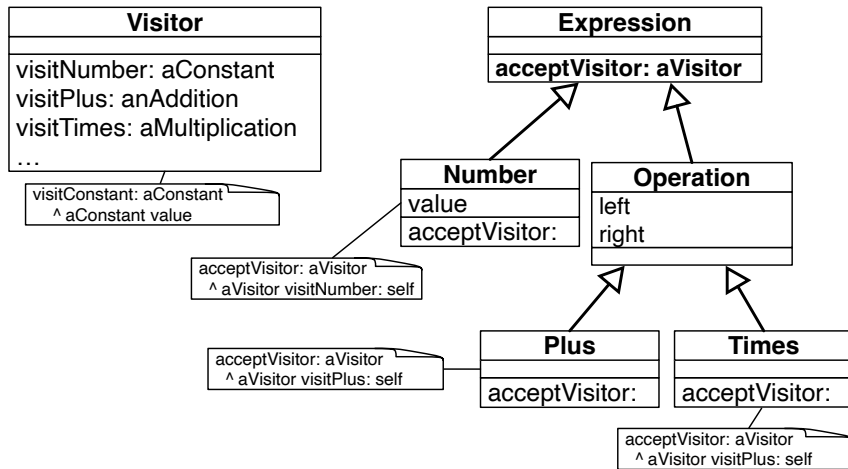


# Visitor's intent

- Represents an **operation** to be performed on the elements of an object structure in a class **separate** from the elements themselves.
- Visitor lets you define a new operation **without changing the classes of the elements** on which it operates.



# Visitor's design



# Visitor + Composite

A visitor requires a structure to perform different actions based on the kind of element

- Perfect match with a composite
- Uses double dispatch

A visitor separates a treatment from the data structure (Composite) it applies to.



## Example: Evaluator Visitor

```
Evaluator >> visitNumber: aNumber  
  ^ aNumber value
```

```
Evaluator >> visitPlus: anExpression  
  | l r |  
  l := anExpression left acceptVisitor: self.  
  r := anExpression right acceptVisitor: self.  
  ^ l + r
```

```
Evaluator >> visitTimes: anExpression  
  | l r |  
  l := anExpression left acceptVisitor: self.  
  r := anExpression right acceptVisitor: self.  
  ^ l * r
```



# Invoking the Visitor

Evaluator new evaluate:

(Plus

left: (ENumber value: 1)

right: (Times left: (ENumber value: 3) right: (ENumber value: 2)))

> 7

Evaluator >> evaluate: anExpression

^ anExpression acceptVisitor: **self**



## Example: Printer

Visitor subclass: #Printer  
iv: 'stream level'

Printer >> visitNumber: aNumber  
stream nextPutAll: aNumber value asString

Printer >> visitPlus: anExpression  
stream nextPutAll: '+'.  
anExpression left acceptVisitor: self.  
anExpression right acceptVisitor: self.

Printer >> visitPlus: anExpression  
stream nextPutAll: '\*'.  
anExpression left acceptVisitor: self.  
anExpression right acceptVisitor: self.





# Expression Visitor analysis

- Each visitor **knows** what to do for a number, a plus, and times operation
- Each visitor manages its **own specific** state
- Each visitor is **independent** of other ones
- Double dispatch supports the decoupling



# A protocol for extension

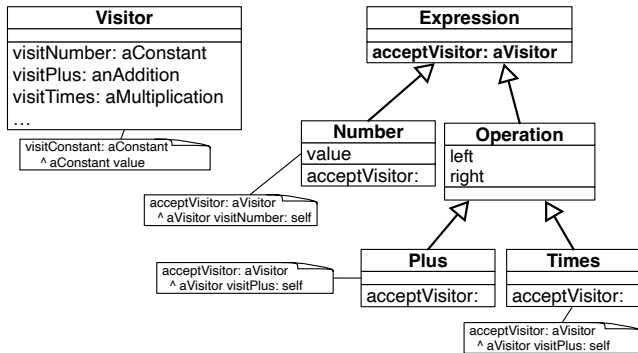
To extend a visitor:

- Define a class which has the expected API i.e., visitX methods
- Apply the visitor to the structure



# Stepping back: double dispatch is the key point

- The Visitor knows the elementary operations (e.g., evaluating a plus, a minus, and a value)
- The items mentions to the Visitor how **they** want to be visited



# Visitor: another look at it

Visitor design provides a **pluggable distributed recursive treatment** of a composite structure

```
Printer >> visitPlus: anExpression  
  stream nextPutAll: '*' .  
  anExpression left acceptVisitor: self.  
  anExpression right acceptVisitor: self.
```



# Stepping back: a basic trolling question

Some people may tell you that Visitor is not OO because Visitor externalizes behavior out of objects.

- Yes operations applied on objects are defined outside the objects.
- Let us think:
  - How much extra behavior and extra state **mixed for nothing** are you ready to pay?
  - Do you **want to package multiple** behaviors **separately**?
- If you have a lot of orthogonal treatments, then better separate them



# When to use a Visitor

- Whenever you have a number of items on which you have to perform a number of actions

Examples:

- Parse tree (ProgramNode) uses a visitor for
  - the compilation (emitting code on CodeStream),
  - pretty printing, syntax highlighting
  - different analysis pass,
  - rotten green test analysis
- Rendering documents (Document) in different formats
  - nodes expansion, HTML, LaTeX, ...



# When using a Visitor is challenging

Changing node elements

- If the elements of the composite change, you will have to change all your visitors
- Problem known as the expression problems in statically-typed languages



# Conclusion

Pros:

- Visitor is a good pattern
- It provides modular and extensible design
- Double dispatch makes it plug and play

Cons:

- It is complex
- It is not adapted to changing structures





A course by

S. Ducasse, L. Fabresse, G. Polito, and Pablo Tesone



Except where otherwise noted, this work is licensed under CC BY-NC-ND 3.0 France

<https://creativecommons.org/licenses/by-nc-nd/3.0/fr/>