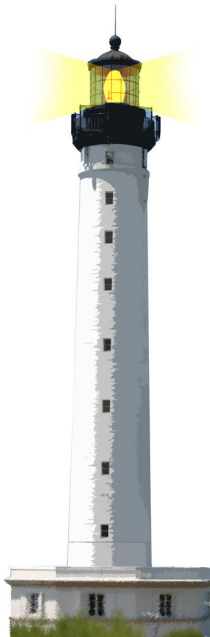
Some Visitor advanced points

S. Ducasse



Pharo

http://www.pharo.org

Goals

Let us chew a bit more Visitor

- What about navigation control
- About better hooks
- Not shortcutting double dispatch

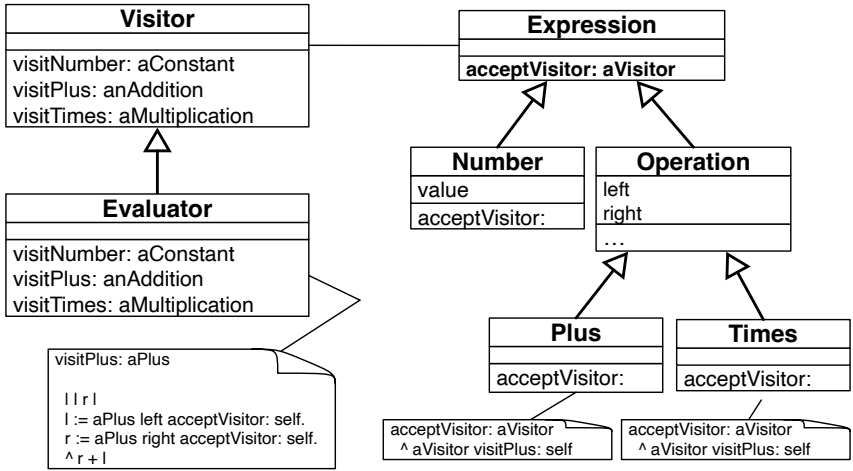
Controlling the traversal

A visitor embeds a structure traversal

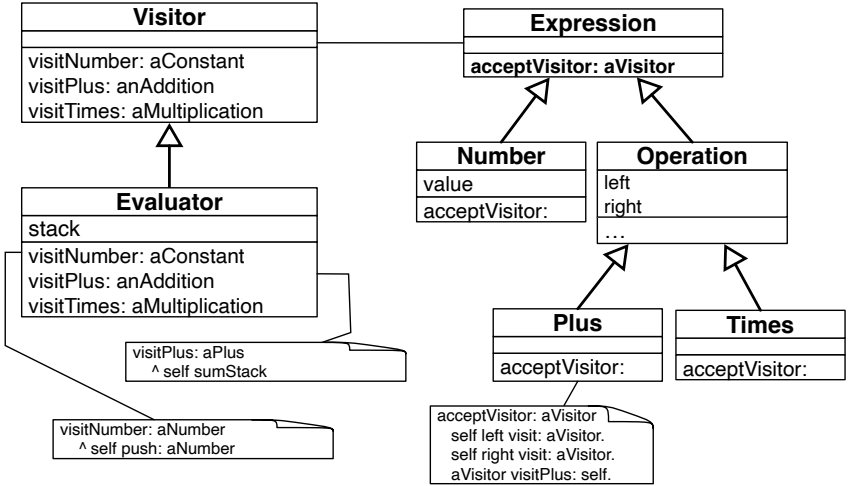
- There are different places where the traversal can be implemented:
  - in the visitors
  - in the items themselves

Usually the visitor is under control but may be the domain logic is more important.

Visitor in control



Items in control



Visitor vs. class extension

Even if a language supports class extension (defining methods on a class from another package than the class package), using a Visitor is better because:

- Each Visitor encapsulates a complex operation
- Each Visitor has its own state

A basic trolling point

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

VisitMethods encode a context

- The granularity of visit methods has an impact on the hooks they offer
- **visit*** methods can be used to provide context

Example: visitTemporariesNode: vs. visitNode:

Compare

```
RBProgramNodeVisitor >> visitSequenceNode: aSequenceNode
  aSequenceNode temporaries do: [:each | self visitNode: each ].
  aSequenceNode statements do: [ :each | self visitNode: each ]
```

vs.

```
RBProgramNodeVisitor >> visitSequenceNode: aSequenceNode
  self visitTemporaryNodes: aSequenceNode temporaries.
  aSequenceNode statements do: [ :each | self visitNode: each ]

RBProgramNodeVisitor >> visitTemporaryNodes: aNodeCollection
  ^ aNodeCollection do: [ :each | self visitTempDefinitionNode: each ]

RBProgramNodeVisitor >> visitTempDefinitionNode: aNode
  ^ aNode acceptVisitor: self
```

Example: visitTemporariesNode: vs. visitNode:

Compare

```
RBProgramNodeVisitor >> visitSequenceNode: aSequenceNode
  aSequenceNode temporaries do: [:each | self visitNode: each ].
  aSequenceNode statements do: [ :each | self visitNode: each ]
```

vs.

```
RBProgramNodeVisitor >> visitSequenceNode: aSequenceNode
  self visitTemporaryNodes: aSequenceNode temporaries.
  aSequenceNode statements do: [ :each | self visitNode: each ]
```

- visitTemporaryNodes: encodes the fact that it is only invoked on temporaries
- No need to guess by looking at parent or other information

About short cutting double dispatch

Imagine that we have the following:

```
RBProgramNodeVisitor >> visitSequenceNode: aSequenceNode
  self visitTemporaryNodes: aSequenceNode temporaries.
  aSequenceNode statements do: [ :each | self visitNode: each ]

RBProgramNodeVisitor >> visitVariable: aNode
  ^ aNode
```

vs.

```
RBProgramNodeVisitor >> visitSequenceNode: aSequenceNode
  self visitTemporaryNodes: aSequenceNode temporaries.
  aSequenceNode statements do: [ :each | self visitVariable: each ]
```

In the second version, the use of visitVariable: aNode

- we are short cutting the double dispatch
- we are cutting the possibility of letting any object participates by telling the visitor how to handle it

Building generic Visitors can be difficult

- Should we return always a result?
- Should collect the values on collection?
- Often the solution is to have an abstract visitor and to redefine most of the logic per family of tasks

Should we promote collections as domain nodes?

- When we iterate on a collection (e.g. of nodes), the collection is not part of the composite domain
- Should we turn such a collection into a domain element?
- It depends of the domain
- and if the it Is the benefit

[Type] Do not overload ==visit== methods

As a summary, overloading does not really work in Java and you will have to explicitly cast your visitor or use getClass everywhere.

- Better define method visitNumber(), visitPlus(), visitTimes()
- than visit()
- Static type may prevent subclass redefinitions to be invoked

Trust an expert :)

Conclusion

- Visitor can be tricky to master
  - use accept/visit vocabulary to really help you
- Visitor is nice for complex structure operations

A course by

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