**Visitor**

Represent 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 Intent**

*Intent:* Represent 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.
**When to use a Visitor**

Whenever you have a number of items on which you have to perform a number of actions, and when you 'decouple' the actions from the items.

Examples:
- the parse tree (ProgramNode) uses a visitor for the compilation (emitting code on CodeStream)
- GraphicsContext is a visitor for VisualComponents, Geometrics, and some other ones (CharacterArray, ...)
- Rendering documents

**Applying the Visitor**

So all our problems are solved, no?
Well...
- when to use a visitor
- control over item traversal
- choosing the granularity of visitor methods
- implementation tricks

**When to Use a Visitor**

Use a Visitor:
- when the operations on items change a lot.

Do not use a visitor:
- when the items you want to visit change a lot.

Question: But how do we know what to choose up-front?

**Visitor Toy Example**

Language to deal with arithmetic expressions.
It supports one kind of number, and has +, *, ( )
We want to evaluate expressions, and print them.
Example:

```
1 + 1
result: 1 + 1 = 2
((4 * 2) * (3 + 5)) * 3
result: (4 * 2 * (3 + 5)) * 3 = 192
```

...
Visitor Toy Example: ParseTree

Two solutions:
- add methods for evaluating, printing, ... on Expression and its subclasses
- create a Visitor, add the visit methods on Expression and its subclasses, and implement visitors for evaluation, printing, ...

Visitor Toy Example Solution 1

Visitor Toy Example 2
Toy Example: Discussion

So which solution to take?
In this case you might say:
  printing is not easy
  adding it directly on Expression clutters Expression (need
to add instance variables etc.)
  therefore we can factor out the printing on a separate
class.
  if we do this with a visitor we can then implement
evaluation there as well.

Smalltalk’s class extensions

Smalltalk has class extensions:
  method addition
  method replacement
So ‘Decoupling’ actions from items can be done:
e.g., put all the printing methods together.
take care: works only for methods
makes it also really easy to package a visitor!
Note: this is a static solution!

Controlling the traversal

Somewhere in the visitor, items are traversed.
Different places where the traversal can be
implemented:
  in the visitor
  on the items hierarchy

Traversal on the Visitor

Smalltalk has class extensions:
  method addition
  method replacement
So ‘Decoupling’ actions from items can be done:
e.g., put all the printing methods together.
take care: works only for methods
makes it also really easy to package a visitor!
Note: this is a static solution!
Traversal on the Items

Visitor
- self visit: aVisitor.
- self right visit: aVisitor.
- aVisitor visitPlus: self.

- self self visit: aVisitor.
- self left visit: aVisitor.
- self visitPlus: self.

Granularity of Visit Methods

Sometimes you can pass context information with the visit methods
So visitors have more information for implementing their operations

Granularity of Visit Methods

- Whole acceptVisitor: self
- ProgramNode acceptVisitor: self
- ProgramNodeEnumerator acceptVisitor: self
- VariableNode acceptVisitor: self
- SequenceNode acceptVisitor: self
- "=do: (each) self (doNode: each)"
- "do: (doNode: each)"
- "do: (doNode: each)"
- "do: (doNode: each)"
- "do: (doNode: each)"

Refined Granularity

Here methods allow finer control of variables
(#doTemporaryVariable)
Implementation Tricks

You can implement it as we have shown before.
But notice the general structure of the methods!
This can be taken as advantage:
  code can be generated for a visitor.
  the method can be performed/invoked
But take care:
  only works when there is a full correspondence.
  can make the code hard to understand.

Using #perform: