Code Generation Witchcraft

And Some Pharo Visitor Stuff

Pierre Misse-Chanabier

Visitors
And Abstract Syntax Tree (AST)

aMethod
   collection do:
      [:anArgument |
        anArgument doStuff]

visitBlockNode: aBlockNode
  self visitArgumentNodes: aBlockNode arguments. ⟶ [:anArgument |
  self visitNode: aBlockNode body ⟶   anArgument doStuff]

Visitors

AST Visitors Are Everywhere

- Compiler

- Code Styler

- Parse Tree Searcher

- Code Formator

- ~50 visitor **subclasses** in the base image

Visitors
Example of **OCScopesCollector**

- 2 Test classes as users

- 4 **very** simple methods

- Do we really need to reify it?

Anonymous Visitor
The Visitor Pattern Without Creating Classes

```
visitor := RBAnonymousVisitor new.
counter := 0.
visitor registerBlock: [ :aNode | counter := counter + 1 ]
    for: #messageNode.
aMethod acceptVisitor: visitor.
```

https://github.com/hogoww/AnonymousVisitor

Anonymous Visitor
Add-hoc Visitor Reification

```
visitor := RBAnonymousVisitor new.
counter := 0.
visitor registerBlock: [ :aNode | counter := counter + 1 ]
    for: #messageNode.
aMethod acceptVisitor: visitor.

visitor reifyAs: #MessageNodeCounterVisitor
```

https://github.com/hogoww/AnonymousVisitor

Visitor Generator
Basic Visitors for Any Hierarchy

VisitorsGenerator generateForRootClass: YourHierarchyClassRoot

- Abstract Visitor

- SubclassResponsibility Visitor

- Superclass Visitor

- Anonymous Visitor ?

- Trait Versions ?

https://github.com/hogoww/VisitorGenerator

Pharo Code Generation
Basic

```
PharoDays compile:
'initialize
    collection := OrderedCollection new
    ''Initializing done !'' crTrace'.
```

Pharo Code Generation
Format

```
instVarName := #collection.
PharoDays compile: (
'initialize
    {1} := OrderedCollection new
    ''Initializing done !'' crTrace'
    format: { instVarName })
```

Pharo Code Generation
Format but Better

```
instVarName := #collection.
PharoDays compile: (
'initialize
    {aCollection} := OrderedCollection new
    ''Initializing done !'' crTrace'
    format: { #aCollection -> instVarName } asDictionary)
```

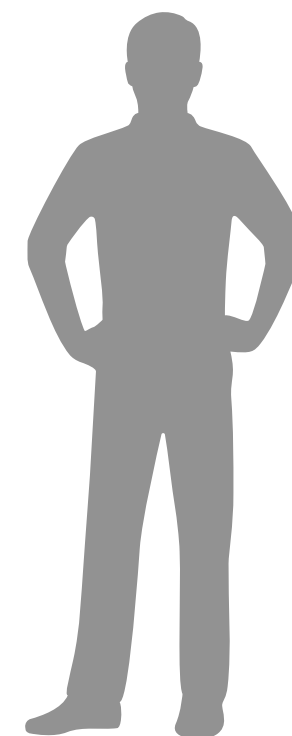# Pharo Code Generation

With a Stream

```
aCollectionName := #collection.
methodBody := String streamContents: [ :stream |
    stream << 'initialize' ; cr.
    stream << aCollectionName << ':='
            << 'OrderedCollection new' ; cr.
    stream << '''Initializing done !'' crTrace'].
PharoDays compile: methodBody.
```

Plain Pharo Code Generation
How About With a Compiler ?

```
aCollection := #collection.
sourceAST := #initialize asMethodWithBody: [
    aCollection := OrderedCollection new.
    'Initializing done !' crTrace.
    ]
PharoDays compile: sourceAST printString.
```

Some Colleague

It's kinda cool,
But it's witchcraft !

https://github.com/hogoww/PlainPharoCode

Plain Pharo Code Generation
In a Nutshell

Used as replacement

Here

aCollection := #collection.

sourceAST := #initialize asMethodWithBody: [

    aCollection := OrderedCollection new.

    'Initializing done !' crTrace.

    ]

PharoDays compile: sourceAST printString.

Never executed

Parsing

https://github.com/hogoww/PlainPharoCode

Plain Pharo Code Generation
Why is it Witchcraft?

https://github.com/hogoww/PlainPharoCode

Plain Pharo Code Generation

Advantages

+ Compiler checks

+ Contextual variable expansions

+ More readable

+ Witchcraft

- Pharo Implementation dependent

- Image tooling (Code styling, Critics…)

https://github.com/hogoww/PlainPharoCode

Plain Pharo Code Generation

Image Tooling

```smalltalk
methods do: [ :m |
    | visitMethodName nodeKind methodSource |
    visitMethodName := m selector.
    nodeKind := (visitMethodName last: visitMethodName size - 5) uncapitalized.
    nodeKind := nodeKind first: nodeKind size - 1.
    register add: nodeKind.
    methodSource := visitMethodName asMethodWithBody: [ :aKindOfNode |
        visits at: #aKindOfNode ifPresent: [ :aBlock | aBlock cull: aKindOfNode ].
        ^ super visitMethodName: { aKindOfNode }
    ] withArguments: { #aKindOfNode -> nodeKind } asDictionary.
    [ RBAnonymousVisitor compile: methodSource asString classified: m protocol ] on: Exception do: [:e| e pass ]
].

register := register asArray sort.
```

12/42 [17]

⊘ [visitMethodName:] Super and Self Messages sent but not implemented ✕ ?
⚠ Long methods ✕ ?
⚠ Sends different super message ✕ ?
⚠ Eliminate unnecessary not's ✕ 🔧 ?
⊘ [visitMethodName:] Messages sent but not implemented ✕ ?
⚠ Uses do: instead of collect: or select:'s ✕ ?
⚠ Rewrite super messages to self messages ✕ 🔧 ?
⚠ Uses do: instead of collect: or select:'s ✕ ?

https://github.com/hogoww/PlainPharoCode

Plain Pharo Code Generation
Compiler Checks

```
PharoDays compile:
'initialize
    collection := OrderedCollection new
    ''Initializing done !'' crTrace'.
```

```
instVarName := #collection.
PharoDays compile: (
'initialize
    {1} := OrderedCollection new
    ''Initializing done !'' crTrace'
format: { instVarName })
```

```
instVarName := #collection.
methodBody := String streamContents: [ :stream |
    stream << 'initialize' ; cr.
    stream << tempName << ':='
            << 'OrderedCollection new'; cr.
    stream << '''Initializing done !'' crTrace'].
PharoDays compile: methodBody.
```

Plain Pharo Code Generation
Variable Expansion in Messages

```
initializerMethods := #(initializeServer initializeClient initializeUsers).
sourceAST := #initializeEverything
        asMethodWithBody: [ self initializerMethods ].
```

↓

initializeEverything
    self initializeServer.
    self initializeClient.
    self initializeUsers

https://github.com/hogoww/PlainPharoCode

Plain Pharo Code Generation
Variable Expansion in Literals

```
kindsOfNodes := self computeKindsOfNodes.
sourceAST := #kindOfNodeExists:
    asMethodWithBody: [ :aKindOfNode |
        ^ #(kindsOfNodes) includes: aKindOfNode  ].
```

⬇

```
kindOfNodeExists: aKindOfNodes
    ^ #( 'argumentNode' 'argumentNodes' 'argumentVariableNode' 'arrayNode' 'assignmentNode' 'blockNode' 'cascadeNode'
    'classVariableNode' 'englobingErrorNode' 'globalNode' 'globalVariableNode' 'instanceVariableNode'
    'literalArrayNode' 'literalNode' 'literalValueNode' 'literalVariableNode' 'localVariableNode' 'messageNode'
    'methodNode' 'node' 'parseErrorNode' 'patternBlockNode' 'patternWrapperBlockNode' 'pragmaNode' 'returnNode' 'selectorNode'
    'selfNode' 'sequenceNode' 'slotInitializationNode' 'storeIntoTempNode' 'storePopIntoTempNode' 'superNode'
    'temporaryDeclarationNode' 'temporaryNode' 'temporaryNodes' 'temporaryVariableNode' 'thisContextNode' 'unreachableStatement'
    'variableNode' ) includes: aKindOfNode
```

19

https://github.com/hogoww/PlainPharoCode

Plain Pharo Code Generation
User Projects

- Anonymous Visitors (https://github.com/hogoww/AnonymousVisitor)

- Visitor Generator (https://github.com/hogoww/VisitorGenerator)

- C-AST (https://github.com/hogoww/C-AST/)

https://github.com/juliendelplanque/PharoCodeGenerator

Conclusion

- Anonymous Visitors !

- We can use blocks to generate Pharo Code

- Pharo Witchcraft

```
visitor := RBAnonymousVisitor new.
counter := 0.
visitor registerBlock: [ :aNode | counter := counter + 1 ]
        for: #messageNode.
aMethod acceptVisitor: visitor.


visitor reifyAs: #MessageNodeCounterVisitor
```

```
aCollection := #collection.
sourceAST := #initialize asMethodWithBody: [
        aCollection := OrderedCollection new.
        'Initializing done !' crTrace.
        ]
PharoDays compile: sourceAST printString.
```

Pierre Misse-Chanabier
pierre_misse25@msn.com
github.com/hogoww
Discord tag: hogo#8547