Slot Composition

Marcus Denker
Part 1: Introduction
Everything is an Object
Everything?
What about Variables?
Object subclass: #Point
    instanceVariableNames: 'x y'
    classVariableNames: ''
    package: 'Kernel-BasicObjects'
Instance Variables

Class Variable

They are not objects!
Object subclass: #Point
  instanceVariableNames: 'x y'
  classVariableNames: ''
  package: 'Kernel-BasicObjects'

This is just a String!
Instance Variables
Class Variables

They are not objects!
Not just definition, the whole reflective API is string / offset based!
Point instVarNames

5@6 instVarAt: 1

5@6 instVarAt: 1 put: 2

5@6 instVarNamed: #x put: 2
We can do better!
We did do better!
Slots and ClassVariables in Pharo
Slots in 2 minutes
Slots: First Class Ivars

- Every instance variable definition is described as an instance of Slot (or subclass)
- Compiler delegates code generation to Slot class
- You can easily make your own!
- A set of interesting Slots are emerging
Slots: First Class Ivars

Object subclass: #Point
slots: { #x. #y }
classVariables: { }  
package: 'Kernel-BasicObjects'

• For InstanceVariableSlot: we write just the #name

• Bytecode exactly the same as ST80 Instance Variables
Slots: API

pointXSlot := Point slotNamed: #x.

#we can read
pointXSlot read: (4@5).
pointXSlot write: 7 to: (4@5).

pointXSlot usingMethods.
pointXSlot astNodes.
pointXSlot assignmentNodes.
Slots: make your own

Slot subclass: #ExampleSlotWithState
  slots: { #value }
  classVariables: { }
  package: 'Slot-Examples-Base'

read: anObject
  ^ value

write: aValue to: anObject
  value := aValue
Slots: First Class Ivars

Object subclass: #MyClass
slots: { #ivar => ExampleSlotWithState }
classVariables: { }
package: 'Kernel-BasicObjects'

- we can compile normal read and assignment
- state ends up in the slot (inspect the slot!)
Slots: more…

• **bytecode**: override `#emitStore:` and `#emitValue:`

• **class builder** calls `#installingIn:` on class creation

• **Initialize instances**: if `#wantsInitialization` is true, `#new` sends `#initialize:` to all slots with the new instance as parameter

• Slots can be **invisible** (just implement `#isVisible`)
Examples

• PropertySlot
• BooleanSlot
• UnlimitedInstanceVariableSlot
• HistorySlot
• ProcessLocalSlot
• ComputedSlot

• RelationSlot
• LazySlot
• InitializedSlot
• ComputedSlot
• SpObservableSlot
• WriteOnceSlot
Start to be used

- In Pharo:
  - Spec: Observable Slot

- Others:
  - Famix: relations, meta data (tag)
  - Typed Slots Project
Part 2: The Composition Problem
Let’s take just two

- SpObservableSlot
- Slot with a default value (InitializedSlot or LazySlot)
Let’s take just two

Object subclass: #MyClass2
  slots: { #ivar => LazySlot default: 5 }
  classVariables: { }
  package: ‘Kernel-BasicObjects’

Object subclass: #MyClass2
  slots: { #ivar => SpObservableSlot }
  classVariables: { }
  package: ‘Kernel-BasicObjects’
I want a SpObservableSlot with default value!
What to do now?
I could implement LazyObservableSlot
Combinatorial Explosion

- PropertySlot
- BooleanSlot
- UnlimitedInstanceVariableSlot
- HistorySlot
- ProcessLocalSlot
- ComputedSlot
- RelationSlot
- LazySlot
- InitializedSlot
- ComputedSlot
- SpObservableSlot
- WriteOnceSlot
SpObservableSlot + LazySlot default: 5
Let's take just two

Object subclass: #MyClass2
  slots: { #iv => SpObservableSlot + LazySlot default: 5 }  
classVariables: {}  
package: 'Examples'
It is not that simple

• We want to compose *instances*, not classes!

• We want to **combine** behaviour: e.g. three slots want to change what happens after a read

• Inheritance or Traits do not solve the problem
Kind of Slots

- Storage ("Implementation")
- Decorators
- Wrappers
Storage

- Define how to store data.
- Examples: PropertySlots, InstanceVariableSlot, ComputedSlot
- It only makes sense to have one
Decorators

- Before / After read and write
- Initialize instances: slot gets notified on #new
- class builder hook on class creation
- Meta Data (e.g. tagged slot)
- We can combine as many decorators as we want!
Wrappers

- Wrap and write, unwrap on read
- Example: Weak Slot: Wrap into a Weak Array
- We have some of them, as they are simple to write:
  - Weak, ProcessLocal, History, WriteOnce
Wrappers: Problems

- Turns Write into a read on the outer slot.
- Can not compose easily (order!)
- **Weak + WriteOnce**: the history collection is weak!
- **WriteOnce + Weak**: Write goes to weak array
Wrappers

- Most (all?) wrappers can be implemented as decorators + additional hidden state.
- Let’s support one wrapper for now
Solution

• ComposedSlot: a Slot composed of
  • One Storage Slot. InstanceVariableSlot is default
  • 0..n Decorators:
  • one Wrapper
InstanceVariable +
InitializedSlot default: 5
InitializedSlot default: 5
Reflective Read

read: anObject
    | value |
    self decorators do: [ :decorator |
            decorator beforeRead: anObject ].

value := self wrapperOrImplementor read: anObject.

self decorators reversDo: [ :decorator |
        decorator afterRead: anObject ].

^value.
Status

- Work in progress implementation
- Currently re-implement all existing Slots to be composable
Future Work

- Integrate with Pharo
- Use in Spec: SpObservable + Initialized, for example
- Unify variable definition for Class Variables and Slots
Questions?

InstanceVariable + InitializedSlot default: 5