

Elements of Design - Unit of Reuse

Stéphane Ducasse
 stephane.ducasse@inria.fr
<http://stephane.ducasse.free.fr/>

S.Ducasse

1



Methods are Units of Reuse

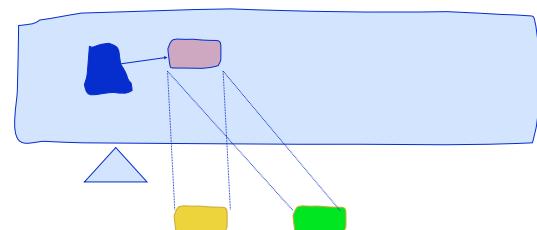
Dynamic binding + methods
 = reuse in subclasses

S.Ducasse

2



Methods are Unit of Reuse



S.Ducasse

3



Example: Forced to Duplicate!

```

Node>>computeRatioForDisplay
| averageRatio defaultNodeSize |
averageRatio := 55.
defaultNodeSize := mainCoordinate / maximiseViewRatio.
self window add:
  (UINode new with:
    (bandWidth * averageRatio / defaultWindowSize)
...
• We are forced to copy the complete method!
SpecialNode>>computeRatioForDisplay
| averageRatio defaultNodeSize|
averageRatio := 55.
defaultNodeSize := mainCoordinate + minimalRatio /
maximiseViewRatio.
self window add:
  (UINode new with: (self bandWidth * averageRatio /
defaultWindowSize))

```

S.Ducasse

4

Self sends: Plan for Reuse



```
Node>>computeRatioForDisplay
| averageRatio defaultNodeSize |
averageRatio := 55.
defaultNodeSize := self defaultNodeSize.
self window add:
    (UINode new with:
        (bandWidth * averageRatio /
defaultWindowSize)
    Node>>defaultNodeSize
        ^ mainCoordinate / maxiViewRatio
SpecialNode>>defaultNodeSize
    ^ mainCoordinate+minimalRatio/maxiViewRatio
```

S.Ducasse

5

Do not Hardcode Class Names



```
Node>>computeRatioForDisplay
| averageRatio defaultNodeSize |
averageRatio := 55.
defaultNodeSize := mainWindowCoordinate / maximiseViewRatio.
self window add:
    (UINode new with:
        (bandWidth * averageRatio / defaultWindowSize)).
    ...
    • We are forced to copy the method!
SpecialNode>>computeRatioForDisplay
| averageRatio defaultNodeSize |
averageRatio := 55.
defaultNodeSize := mainWindowCoordinate / maximiseViewRatio.
self window add:
```

```
(ExtendedUINode new with:
    (bandWidth * averageRatio /
```

S.Ducasse

6

Class Factories



```
Node>>computeRatioForDisplay
| averageRatio |
averageRatio := 55.
self window add:
    self UIClass new with:
        (self bandWidth * averageRatio / self
defaultWindowSize)
    ...

```

Node>>UIClass
^ UINode

SpecialNode>>UIClass
^ ExtendedUINode

S.Ducasse

7

Hook and Template



S.Ducasse

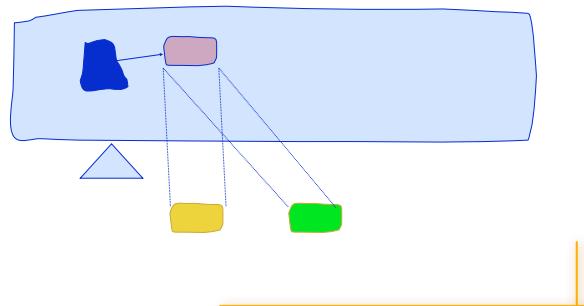
8



Hook and Template Methods



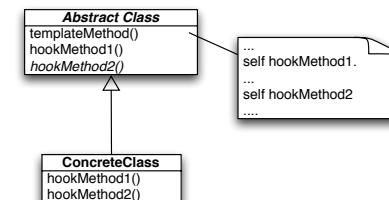
- Hooks: place for reuse
- Template: context for reuse



S.Ducasse

9

Hook and Template Methods



- **Templates:** Context reused by subclasses
- **Hook methods:** holes that can be specialized
- Hook methods do not have to be abstract, they may define default behavior or no behavior at all.

S.Ducasse

10

Hook / Template Example: Printing



Object>>printString

"Answer a String whose characters are a description of the receiver."

```
| aStream |
aStream := WriteStream on: (String new: 16).
self printOn: aStream.
^aStream contents
```

S.Ducasse

11

Hook



Object>>printOn: aStream

"Append to the argument aStream a sequence of characters that describes the receiver."

```
| title |
title := self class name.
aStream nextPutAll:
    ((title at: 1) isVowel ifTrue: ['an '] ifFalse: ['a ']).
aStream print: self class
```

S.Ducasse

12

Overriding the Hook



```
Array>>printOn: aStream  
    "Append to the argument, aStream, the elements of the Array  
     enclosed by parentheses."  
  
| tooMany |  
tooMany := aStream position + self maxPrint.  
aStream nextPutAll: '#'.  
self do: [:element |  
    aStream position > tooMany  
    ifTrue: [ aStream nextPutAll: '...(more)...'.  
        ^self ].  
    element printOn: aStream]  
separatedBy: [aStream space].
```

S.Ducasse

13

Overriding



```
False>>printOn: aStream  
    "Print false."  
  
aStream nextPutAll: 'false'
```

S.Ducasse

14

Specialization of the Hook



The class **Behavior** that represents a class extends
the default hook but still invokes the default one.

```
Behavior>>printOn: aStream  
    "Append to the argument aStream a statement of  
     which  
     superclass the receiver descends from."  
  
    aStream nextPutAll: 'a descendent of '.  
    superclass printOn: aStream
```

S.Ducasse

15

Another Example: Copying



Complex (deepCopy, veryDeepCopy...)
Recursive objects
Graph of connected objects
Each object wants a different copy of itself
No up-front solution

S.Ducasse

16

Hook Example: Copying



Object>>copy

"Answer another instance just like the receiver. Subclasses normally override the postCopy message, but some objects that should not be copied override copy."

`^self shallowCopy postCopy`

Object>>shallowCopy

"Answer a copy of the receiver which shares the receiver's instance variables."

S.Ducasse

17

postCopy



Object>>postCopy

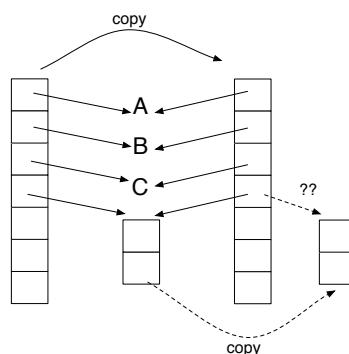
"Finish doing whatever is required, beyond a shallowCopy, to implement 'copy'. Answer the receiver. This message is only intended to be sent to the newly created instance. Subclasses may add functionality, but they should always do super postCopy first."

`^self`

S.Ducasse

18

Sounds Trivial?



S.Ducasse

19

Hook Specialisation



Bag>>postCopy

"Make sure to copy the contents fully."

```
| new |
super postCopy.
new := contents class new: contents capacity.
contents keysAndValuesDo:
[:obj :count | new at: obj put: count].
contents := new.
```

S.Ducasse

20

Guidelines for Creating Template Methods

Simple implementation.

Implement all the code in one method.

Break into steps.

Comment logical subparts

Make step methods.

Extract subparts as methods

Call the step methods

Make constant methods, i.e., methods doing nothing else than returning.

Repeat steps 1-5 if necessary on the methods created

