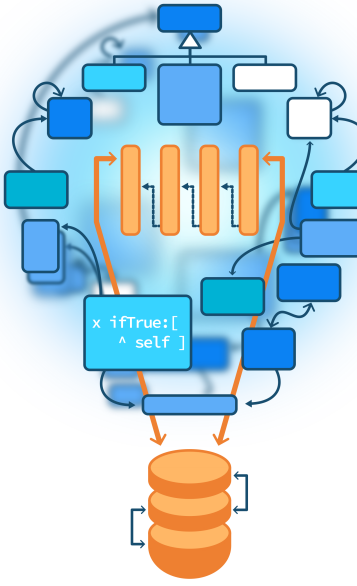
Behavior delegation at work

The case of the class printer

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



Pharo

http://www.pharo.org

Goals

- Think **modular**
- Look at the class definition printer
- Check design in Pharo

Context and challenges

How to support **various** class definition formats in a **modular** way?

- Formats:
  - Old Squeak syntax, old Pharo, and Fluid syntax
- **Different** objects:
  - class, metaclass, trait...
- How to **control** the complexity?
  - Slots should not be displayed in Old Pharo
- How to avoid **checks** everywhere?
- And can we support removing one definition at any time without recompilation

First hacked version in Pharo 70/80

- Introduction of support for slots was hacked
- Smell like duplication

```
ClassDescription >> definition
  (self needsSlotClassDefinition or: [ Slot showSlotClassDefinition ])
    ifTrue: [ ^ self definitionWithSlots ].
  ^ self definitionWithoutSlots
```

```
Metaclass >> definition
  (self slotsNeedFullDefinition or: [ Slot showSlotClassDefinition ])
    ifTrue: [ ^ self definitionWithSlots ].
  ^ self definitionWithoutSlots
```

Hacked in tools too

```
ClyClassCreationToolMorph >> classTemplate

  | template |
  template := Slot showSlotClassDefinition
    ifTrue: [
      'Object subclass: #NameOfSubclass
  slots: {}
  classVariables: {}
  package: ''' ]
    ifFalse: [
      'Object subclass: #NameOfSubclass
  instanceVariableNames: ''''
  classVariableNames: ''''
  package: ''' ].
  ^ template , self packageName , ''''
```

Thinking... about a solution

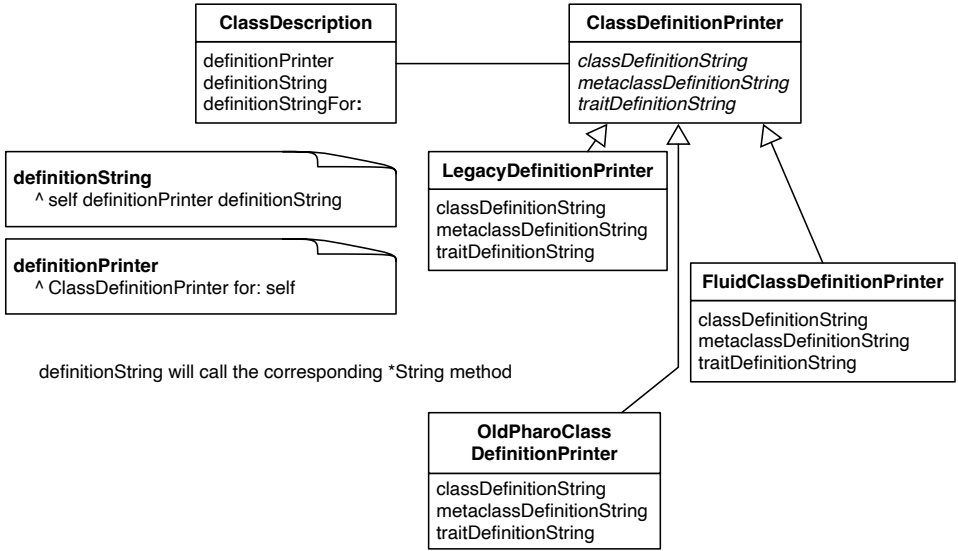
Think 5 min how you would solve it

Solution: Delegating to a class definition printer

- Create class definition printers
- A kind of Strategy Design Pattern
- A hierarchy of printers + a factory + some double dispatch

```
ClassDefinitionPrinter ( forClass )
  FluidClassDefinitionPrinter
  LegacyClassDefinitionPrinter
  OldPharoClassDefinitionPrinter
```

Solution: Delegating to a class definition printer



The class asks a printer

```
ClassDescription >> definitionString
  ^ self definitionPrinter definitionString
```

The printer factory creates and return a printer for the class

```
ClassDescription >> definitionPrinter
  ^ ClassDefinitionPrinter for: self
```

A printer

Knows how to print

- **class** (classDefinitionString)
- **metaclass** (metaclassDefinitionString)
- **trait** (traitDefinitionString)

Fluid class printer at work

A class

```
Object << #Point
    slots: { #x . #y };
    tag: 'BasicObjects';
    package: 'Kernel'
```

A metaclass without class instance variables

```
Object class << Point class
```

Fluid printer: a class

```
FluidClassDefinitionPrinter >> classDefinitionString

  ^ String streamContents: [ :s |
      forClass superclass
        ifNotNil: [ s nextPutAll: forClass superclass name ]
        ifNil: [ s nextPutAll: 'nil' ].
    self msgAndClassNameOn: s.
    ...
    ...
    forClass slots ifNotEmpty: [ self slotsOn: s ].
    forClass classVariables ifNotEmpty: [ self sharedVariablesOn: s ].
    forClass sharedPools ifNotEmpty: [ self sharedPoolsOn: s ].
    self tagOn: s.
    self packageOn: s ]
```

Fluid printer: a metaclass

```
FluidClassDefinitionPrinter >> metaclassDefinitionString

  ^ String streamContents: [ :strm |
    forClass superclass
      ...
        strm
          nextPutAll: forClass superclass name;
          nextPutAll: ' << ';
          nextPutAll: forClass name ]
      ...
    self lastTraitsOn: strm.
    forClass slots ifNotEmpty: [ self lastSlotsOn: strm ] ]
```

Old Pharo cannot display slots

```
OldPharoClassDefinitionPrinter >> classDefinitionString
  ^ forClass needsSlotClassDefinition
      ifTrue: [ (ClassDefinitionPrinter fluid for: forClass) classDefinitionString ]
      ifFalse: [ self basicClassDefinitionString ]
```

If the class has slots then

- ask a fluid class printer to do the job

Tool logic is now simpler

```
ClassCreationToolMorph >> classTemplate

  ^ ClassDefinitionPrinter new
      compactClassDefinitionTemplateInPackage: self packageName
```

Analysis

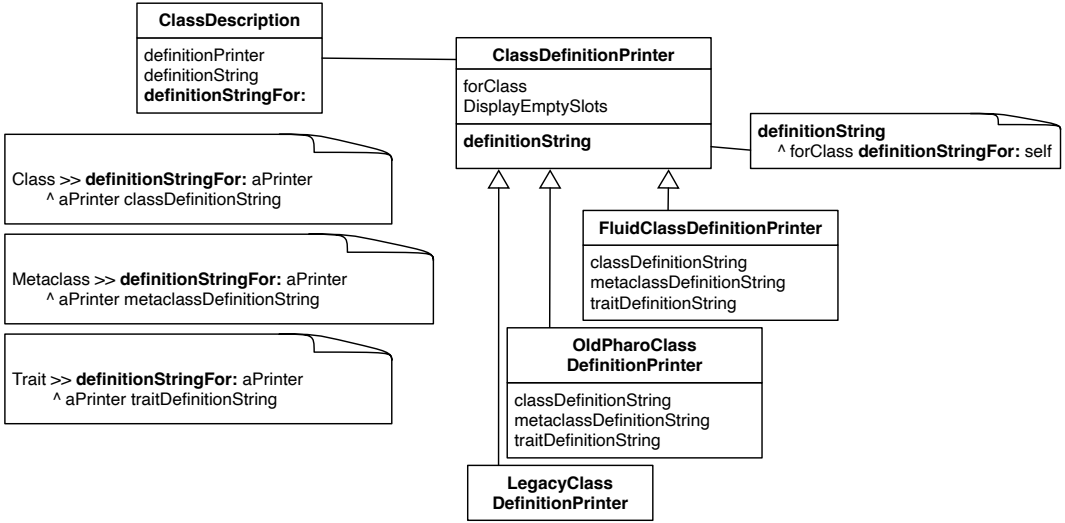
- **Modular**: One format = one printer
- One printer supports **multiple** related features (expansion, template, class printing)
- Supports **reuse** within the hierarchy
- **Defaulting** within printers
  - if necessary legacy printer redirects to fluid printer

More challenges

How to support printing **different** objects on **different** printers?

- Traits, Classes, and Metaclasses
- Legacy, OldPharo, and Fluid

Solution: Delegating to a class definition printer



Double Dispatch

```
ClassDefinitionPrinter >> definitionString
  ^ forClass definitionStringFor: self
```

```
Class >> definitionStringFor: aPrinter
  ^ aPrinter classDefinitionString

Trait >> definitionStringFor: aConfiguredPrinter
  ^ aPrinter traitDefinitionString

TraitedMetaclass >> definitionStringFor: aPrinter
  ^ aPrinter traitedMetaclassDefinitionString
```

More tension

- Do not want to have direct reference to printers to be able to easily **remove** printers in the future

Solution:

- **Limit** reference to specific printers
- Only reference the superclass that acts as a **factory**

Encapsulate printer selection

```
Object << #ClassDefinitionPrinter
  slots: { #forClass };
  sharedVariables: { #DisplayEmptySlots . #ShowFluidClassDefinition };
  tag: 'ClassDefinitionPrinter';
  package: 'Kernel'
```

```
ClassDefinitionPrinter >> new
  ^ self showFluidClassDefinition
      ifTrue: [ self fluid ]
      ifFalse: [ self oldPharo ]
```

Still the possibility to refer to legacy definition

```
ClassDescription >> oldDefinition

  ^ ClassDefinitionPrinter legacy
    for: self;
    definitionString
```

Conclusion

- Dispatch over objects
- Avoid complex conditional cases
- Concentrate object creation point

Produced as part of the course on http://www.fun-mooc.fr

Advanced Object-Oriented Design and Development with Pharo

A course by
S.Ducasse, L. Fabresse, G. Polito, and P. Tesone

Inria
LearningLab

2023