Delegation vs. Inheritance

Basic but worth

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

- Delegation-based and inheritance-based designs
- Compare designs using criteria/hints
Exercise setup

Imagine the class `TextEditor` and the definition of several algorithms:

- `formatWithTeX(t)` to color TeX
- `formatFastColoring(t)` to fastly color the text
- `formatSlowButPreciseColoring(t)` to color ...
- `formatRTF(t)`
- ...

How can we create an editor that will format differently different texts?
Agenda

• Two first solutions:
  ○ with inheritance
  ○ with one class and conditionals
• Define some criteria to compare solutions
• A third solution with delegation
• Evaluation
With inheritance

Object << #TextEditor
  slots: { #text }

TextEditor >> format
  self subclassResponsibility

SlowFormatingTextEditor >> format
  self formatSlowButPreciseColoring: text

FastFormatingTextEditor >> format
  self formatFastColoring: text

NullFormatingTextEditor >> format
  ^ self "do nothing"
With one class and conditionals

<table>
<thead>
<tr>
<th>TextEditor</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
</tr>
<tr>
<td>currentSelection</td>
</tr>
<tr>
<td>formatSlowButPrecise: t</td>
</tr>
<tr>
<td>formatFastColoring: t</td>
</tr>
<tr>
<td>formatWithTex: t</td>
</tr>
</tbody>
</table>

TextEditor >> format
currentSelection = #slow
  ifTrue: [ self formatSlowButPreciseColoring: text]
  ifFalse: [
    currentSelection = #fast
    ifTrue: [self formatFastColoring: text]
    ....
  ]
With one class, a registry and meta-programming

Object << #TextEditor
slots: { #currentSelection. #formatters. #text }

TextEditor class >> initialize
 self formatters
 at: #slow put: #slowFormat: ;
at: #fast put: #fastFormat: ;
at: #null put: #nullFormat: ;
at: #tex put: #texFormat:

TextEditor >> format
 self perform: (formatters at: currentSelection) with: text
How to compare solutions?

Some criteria:

- **Addition**
  - What is the cost to define a new formatting algorithm?

- **Packaging**
  - Can I deploy a new formatting algorithm separately from others?

- **Dynamic switch**
  - Can I dynamically switch to another formatting algorithm?
Evaluating inheritance-based solution

Pros:

- Addition: adding a new formatting algorithm is done by subclassing
- Packaging: formatting algorithms are modularised in separate classes

Cons:

- Dynamic switch
  - Have to create the right TextEditor at beginning
  - Difficult to change it dynamically (external references) and we do not want to reopen the text editor
- Addition: combinatorial explosion
Evaluating inheritance-based solution

- Do not want a hierarchy for each text editor features to be multiplied with previous ones (Single/Multi-Pane, completion, grammatical verification, compilation,....)
- API of TextEditor can get large: no clear identification of responsibilities
Evaluating conditionals-based solution

Pros:
- Dynamic switch: we can use a different formatting algorithm dynamically

Cons:
- Addition: adding a version requires to edit and \texttt{recompile} the conditionals
- Packaging: we cannot package a new algorithm separately
Solution with delegation

Imagine a solution using delegation to another object (a formatter)
Delegating to a formatter

myEditor formatter: FastFormatter new.
myEditor format.
myEditor formatter: SlowFormatter new.
Evaluating the delegation to a formatter

Pros:
- Addition: just add a new formatter subclass
- Packaging: formatting algorithms are well modularised in separate classes
- Dynamic switch: just create a new formatter instance and set it in the editor
- Uniform API between the Editors and the Formatters (format:)

Cons:
- The formatter should access the state of the text (i.e. the text, positions... contained in the text editor)
- The API of the TextEditor should be opened to support it

BTW, this is a typical example of the Strategy Design Pattern ;-)
Conclusion

Inheritance
- is about *incremental static* definition
- can lead to static design
- helps defining *abstractions*

Delegation
- brings runtime *flexibility* and modularity

but there’s no such thing as a free lunch!
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

Except where otherwise noted, this work is licensed under CC BY-NC-ND 3.0 France
https://creativecommons.org/licenses/by-nc-nd/3.0/fr/