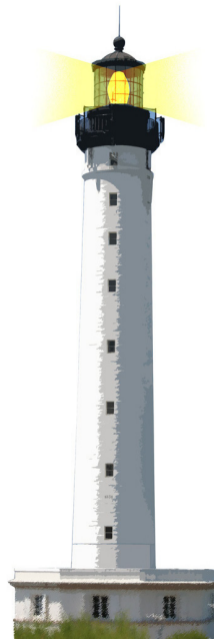


Use vs. Inheritance

Basic but worth

S. Ducasse



Goals

- Compare Use and Inheritance
- Some criteria/hints



Outline

- An exercise
- Some criteria
- Solutions
- Comparing solutions



Exercise setup

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

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

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



Next step

- Propose a solution with inheriting classes
- Propose a solution with one class and conditionals
- Define some criteria & compare
- Propose a solution with delegation
- Compare



With inheritance

```
TextEditor < #SlowFormattingTextEditor
```

```
SlowFormattingTextEditor >> format  
  self formatSlowButPreciseColoring: text
```

```
TextEditor < #FastFormattingTextEditor
```

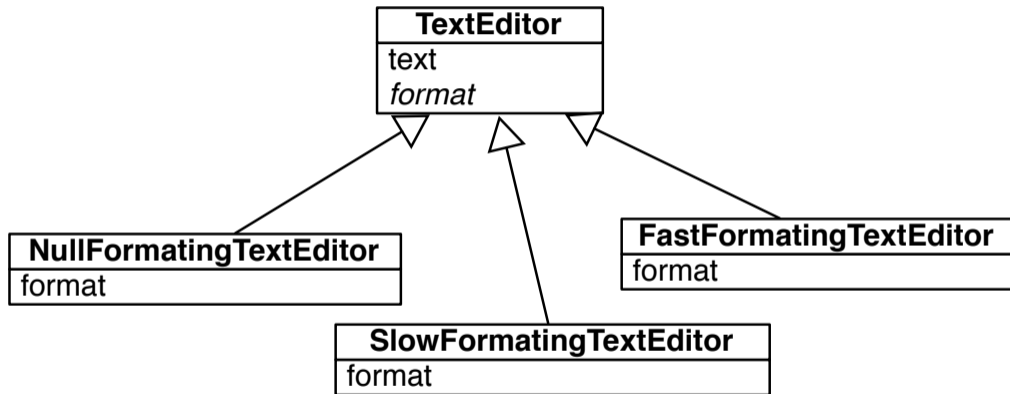
```
FastFormattingTextEditor >> format  
  self formatFastColoring: text
```

```
TextEditor < #NullFormattingTextEditor
```

```
NullFormattingTextEditor >> format  
  ^ self "do nothing"
```



With inheritance



With conditionals

TextEditor
text formatSlowButPrecise: t formatFastColoring: t formatWithTex: t

```
TextEditor >> format
currentSelection = #slow
if True: [ self formatSlowButPreciseColoring: text]
if False: [ currentSelection = #fast
  if True: [self formatFastColoring: text]
  ....]
```


With registry and meta programming

```
Object subclass: #TextEditor  
  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
```

Criteria

- Yes what are they?

Criteria

- **Adding a new formatting algo** - what is the cost to define a new formatting algorithm?
- **Dynamically use a formatter** - can I switch dynamically to a new formatting algorithm?
- **Packaging** - can I deploy a new formatting algorithm separately from others?



Inheritance?

Addition:

- we can add a new formatter

Packaging:

- we can package a new formatter

Not the best solution since:

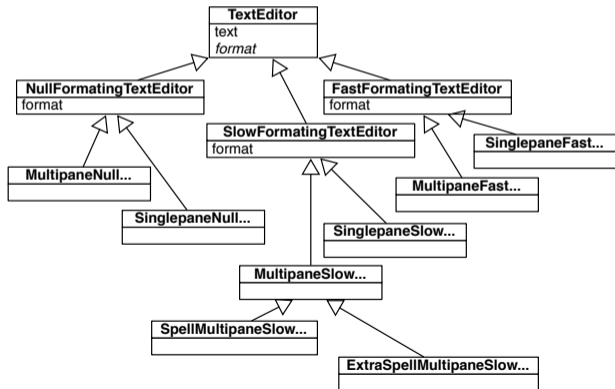
- you have to create objects of the right class
- it is difficult to **change** the policy dynamically.
 - we do not want to have and reopen the texteditor



Inheritance?

You can get an explosion of classes

- we do not want a hierarchy for each text editor features to be **multiplied** with previous ones (imagine completion, grammatical verification, compilation,...)
- API of TextEditor can get large: no clear identification of responsibilities



Conditionals?

Dynamic use: we can use a different formatter dynamically.

But **Addition:**

- adding a version requires to edit and **recompile** the conditionals

Packaging:

- we cannot package a new algorithm separately

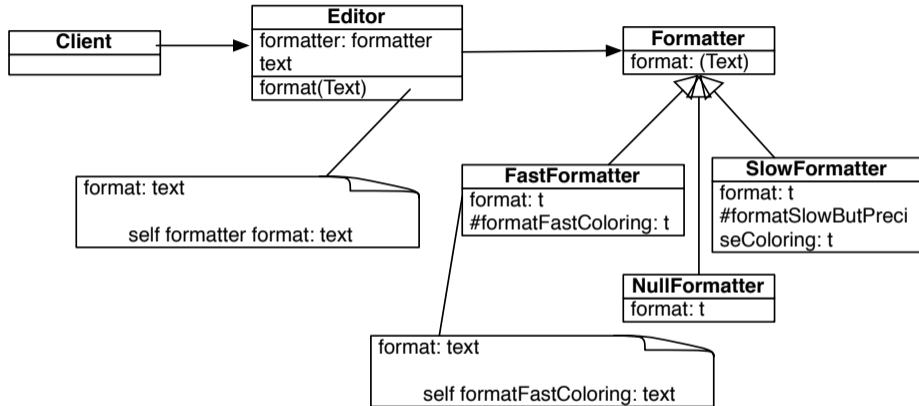


With delegation

Propose a solution using delegation to another object (a formatter)



Delegating to a formatter



```
myEditor formatter: FastFormatter new.  
myEditor format.  
myEditor formatter: SlowFormatter new.
```


Delegating to a formatter

Dynamic use:

- we can use a different formatter dynamically. Just create a new instance and set it.

Addition:

- adding a version is just adding a new class

Packaging:

- we package a new algorithm separately



Strategy Design Pattern

- Uniformize the communication (API) between the Editor and the Formatter
 - all formatters should understand format:
- Modular
- Incremental



There is nothing like a free lunch

- The formatter should access the state of the text (i.e. the text, positions... contained in the text editor)
- Information should **flow** between the textEditor and the formatter
- API of textEditor should be opened to support it



Conclusion

Inheritance

- is about **incremental static** definition
- It can lead of static design
- It help defining **abstractions**

Delegation

- can bring runtime **flexibility** and modularity



A course by

S. Ducasse, G. Polito, and Pablo 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/>