

About Registration

When class method-based registration is too much

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



Goal

- Thinking about system dynamics
- Alternatives to class methods as registration mechanism
- Impact of dynamic registration



Using class methods as registration

- A class is a regular object
- We can send a message to a class
- Each class can answer specifically

Object allSubclasses collect: [:each | each foo]

Each class is able to:

- define its own `foo` method
- reuse the one of its superclass



Remember the previous lecture

```
PillarParser >> documentClasses  
  ^ DocumentItem allSubclasses  
    sorted: [ :class1 :class2 | class1 priority <  
              class2 priority ]
```

```
PillarParser >> parse: line  
  self documentClasses  
    detect: [ :subclass |  
              (subclass canParse: line)  
                ifTrue: [ ^ subclass newFromLine: line ] ]
```



Registration for 'Free'

Pros:

- Each time a new class is loaded it is taken into account

Cons:

- We are querying the system each time
- Most of the time for nothing
- Expensive mechanism



Solution 1: Explicit static list

```
PillarParser >> documentClasses  
^ { Section. List. Paragraph }  
  sorted: [ :class1 :class2 | class1 priority < class2 priority ]
```



Solution 1: Explicit static and ordered list

We could precompute the priority too:

```
PillarParser >> documentClasses  
^ { Section. Paragraph. List }
```



Solution 1: Evaluation

Pros:

- Do not have to query all the classes all the time

Cons:

- You have to keep this list up to date
- Listing explicitly classes may introduce undesired dependencies to other packages!



Solution 2: Explicit registration mechanism

Classes can explicitly register themselves to the parser:

```
PillarParser >> documentClasses  
^ RegisteredClasses
```

```
PillarParser >> registerClass: aDocumentItemClass  
self documentClasses add: aDocumentItemClass
```

```
Section class >> initialize  
PillarParser registerClass: self
```

```
Paragraph class >> initialize  
PillarParser registerClass: self
```



Solution 2: Evaluation

- No need to maintain the list of classes manually
- Dynamic list without querying the system all the time
- Registration could support priority
- External classes can also register

Extra class >> initialize
PillarParser registerClass: **self**

- Do not introduce unwanted dependencies



Unregistration is a concern

Explicit registration requires unregistration.

- The registration holder (here `PillarParser`) should offer way to remove a registration
- Registered classes have the responsibility to unregister themselves (e.g. class unloading)



Conclusion

- MySuperClass subclasses is a cool pattern
 - but it has a cost!
- Better use an explicit registration mechanism
 - it is dynamic and save expensive queries for nothing
- Design is about tradeoffs



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/>