

Sharing with instance specific possibilities

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



Goals

- Thinking about sharing
- How can we share by default a resource?
- How can we share by default a resource and still get **instance-based** usage?



Instance vs. class sharing

Instance specific

- An instance variable (most of the time) holds **instance specific values**

Shared between all instances of a class

- A shared variable (static or class variables) holds a value that is **shared among all instances** of the class



Is it shared or instance specific?

- How can we **share by default** a resource and **still** get **instance-based** use possible?
- Imagine a solution...



Case Study: Scanner

```
> Scanner new scanTokens: '#identifier #keyword: 25 string'  
#(#identifier #keyword: 25 'string')
```



The Scanner class enigma

Imagine the following class:

```
Object << #Scanner
  slots: {#mark . #currentChar . #token . #tokenType . #typeTable};
  sharedVariables: { #TypeTable }
  package: 'Compiler'
```

- Why do we have a shared variable `TypeTable` and an instance variable `typeTable` are defined at the instance
- A bug? - No! This is a nice design
- Do you see it?



Let us explain key aspects

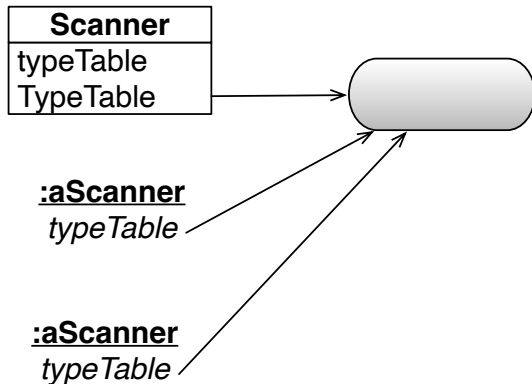
- TypeTable **the** shared variable
 - is initialized **once** to hold the table of elements
 - not used by any instance method
- typeTable **the** instance variable
 - is used by every instance method
 - is initialized by pointing to TypeTable
 - All methods **only** access the instance variable and never the shared one

Do you see the idea?



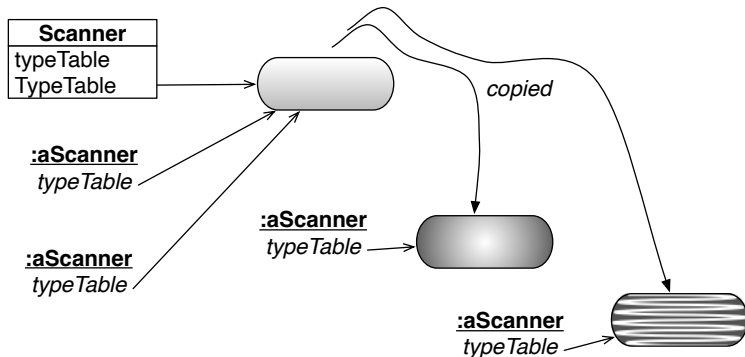
Explanation

- By default all instances share the same type table (a large object)
- All methods can access it via `typeTable`



Specific state for specific instances

- Copy the state of `typeTable` and modify it per instance



- Possible since all methods access instance specific modified state via `typeTable` instance variable

Shared variable points to the share table

```
Scanner class >> initialize
| newTable |
newTable := ScannerTable new: 255 withAll: #default.
newTable atAllSeparatorsPut: #separator.
newTable atAllDigitsPut: #digit.
...
newTable at: $( asInteger put: #leftParenthesis.
newTable at: $^ asInteger put: #upArrow.
...
TypeTable := newTable
```



And...

Instances only access the type table via the instance variable that points to the shared table that has been initialized once.

```
Scanner >> initialize
```

```
  super initialize.
```

```
  typeTable := TypeTable
```



One instance specific state

Scanner new setTypeTable: (Scanner defaultTypeTable copy) customizedForThisUse



A subclass with instance with specific table

A subclass has just to specialize initialize method

```
MyScanner >> initialize  
  super initialize.  
  typeTable := typeTable copy.  
  self modifyTypeTable
```

All the instances of MyScanner will have their own table



Conclusion

- Can get sharing by default
- but get instance specific if need it



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/>