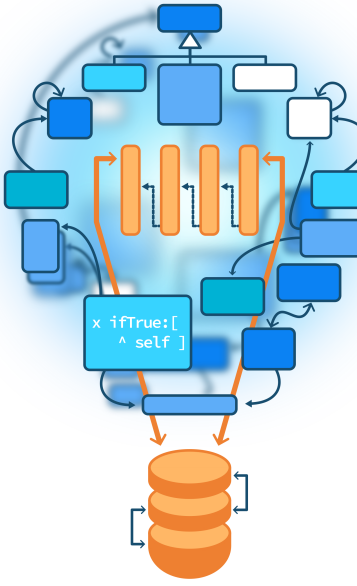
Sharing with instance specific possibilities

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 (not from Pharo)

```
>>> Scanner new scanTokens: 'identifier keyword: 25 embedded.period key:word: . '

  #(#identifier #keyword: 25 'string' 'embedded.period' #key:word: #'.')
```

The Scanner class enigma

```
Object << #Scanner
  slots: {#mark . #prevEnd . #hereChar . #token . #tokenType . #typeTable};
  sharedVariables: { #TypeTable }
  package: 'Scanning'
```

- What? TypeTable and typeTable are defined at the instance and class sharing level. A bug?
- No! This is a nice design
- Do you see it?

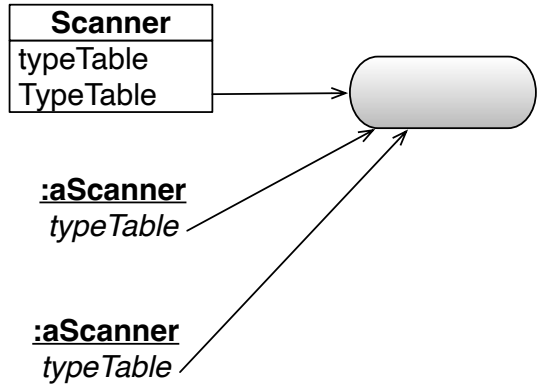
Further investigation

- 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?

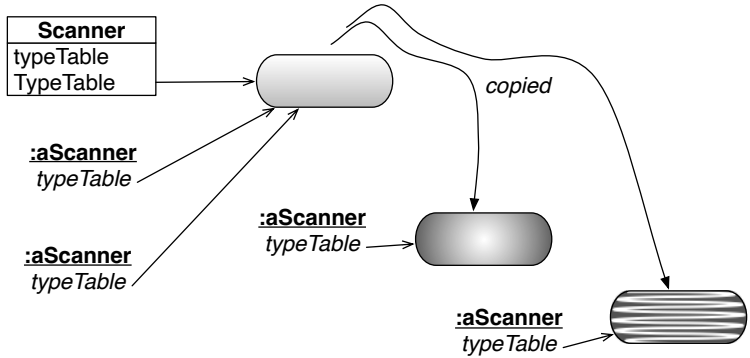
Explanation

- By default all instances share the same typeTable
- All methods can access it via typeTable



Specific state for specific instances

- Copy the state of typeTable and modify it per instance
- 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: #xDefault. "default" newTable
      atAllSeparatorsPut: #xDelimiter.
  newTable atAllDigitsPut: #xDigit.
  newTable atAllLettersPut: #xLetter.
  '!%&*+,-/<=>?@\~' do: [:bin | newTable at: bin asInteger put: #xBinary]. "Other multi-
      character tokens"
  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 class >> new
  ^ super new initScanner

Scanner >> initScanner
  buffer := WriteStream on: (String new: 40). saveComments := true.
  typeTable := TypeTable
```

One instance specific state

```
Scanner new setTypeTable: (Scanner defaultTypeTable copy) andHack
```

A subclass has just to specialize initScanner without copying the initialization of the table.

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

All the instance 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

Inria LearningLab

2023