7. Understanding Classes and Metaclasses
Roadmap

- Metaclasses in 7 points
- Indexed Classes
- Class Instance Variables
- Class Variables
- Pool Dictionaries

Selected material courtesy Stéphane Ducasse
Roadmap

> Metaclasses in 7 points
> Indexed Classes
> Class Instance Variables
> Class Variables
> Pool Dictionaries
Metaclasses in 7 points

1. Every object is an instance of a class
2. Every class eventually inherits from Object
3. Every class is an instance of a metaclass
4. The metaclass hierarchy parallels the class hierarchy
5. Every metaclass inherits from Class and Behavior
6. Every metaclass is an instance of Metaclass
7. The metaclass of Metaclass is an instance of Metaclass

Adapted from Goldberg & Robson, Smalltalk-80 — The Language
Metaclases in 7 points

1. Every object is an instance of a class
2. Every class eventually inherits from Object
3. Every class is an instance of a metaclass
4. The metaclass hierarchy parallels the class hierarchy
5. Every metaclass inherits from Class and Behavior
6. Every metaclass is an instance of Metaclass
7. The metaclass of Metaclass is an instance of Metaclass
1. Every object is an instance of a class
Metaclases in 7 points

1. Every object is an instance of a class
2. Every class eventually inherits from Object
3. Every class is an instance of a metaclass
4. The metaclass hierarchy parallels the class hierarchy
5. Every metaclass inherits from Class and Behavior
6. Every metaclass is an instance of Metaclase
7. The metaclass of Metaclase is an instance of Metaclase
2. Every class inherits from Object

> Every object is-an Object =
  - The class of every object ultimately inherits from Object

@@ Why not ColoredSnake inherits from Snake
Snake inheriting from square Hurts me :_(

Caveat: in Squeak, Object has a superclass called ProtoObject
The Meaning of is-a

> When an object receives a message, the method is looked up in the method dictionary of its class, and, if necessary, its superclasses, up to Object

```
self printOn:
```

I think that this is important
To make a visual distinction
Between message passing and lookup
This is why I had this arrow thingie

```
printString:
```
Responsibilities of Object

> **Object**
  
  -- represents the common object behavior
    
    -- *error-handling, halting* ...
  
  -- all classes should inherit ultimately from **Object**
Metaclases in 7 points

1. Every object is an instance of a class
2. Every class eventually inherits from Object
3. **Every class is an instance of a metaclass**
4. The metaclass hierarchy parallels the class hierarchy
5. Every metaclass inherits from Class and Behavior
6. Every metaclass is an instance of Metaclase
7. The metaclass of Metaclase is an instance of Metaclase
3. Every class is an instance of a metaclass

> Classes are objects too!
> - Every class $x$ is the unique instance of its metaclass, called $x$ class

```
Object
    `__________`
   |         |
   v         v
Square
    `__________`
   |         |
   v         v
Snake
    `__________`
   |         |
   v         v
aSnake
```

Object class

Square class

Snake class

«instanceOf»
Metaclasses are implicit

> There are no explicit metaclasses
  – Metaclasses are created implicitly when classes are created
  – No sharing of metaclasses (unique metaclass per class)
Metaclasses by Example

Square allSubclasses
Snake allSubclasses

Snake allInstances
Snake instVarNames

Snake back: 5

Snake selectors

Snake canUnderstand: #new
Snake canUnderstand: #setBack:

a Set(Snake FirstSquare Ladder)
a Set()
#('back')

←5[nil]

an IdentitySet(#printOn:
    #destination #setBack:)
false
ture
Metaclasses in 7 points

1. Every object is an instance of a class
2. Every class eventually inherits from Object
3. Every class is an instance of a metaclass
4. The metaclass hierarchy parallels the class hierarchy
5. Every metaclass inherits from Class and Behavior
6. Every metaclass is an instance of Metaclass
7. The metaclass of Metaclass is an instance of Metaclass
4. The metaclass hierarchy parallels the class hierarchy
Uniformity between Classes and Objects

> Classes are objects too, so ...
  - Everything that holds for objects holds for classes as well
  - Same method lookup strategy
    - Look up in the method dictionary of the metaclass

@@Is back a class method@@

Object class

Square class

Snake class

Object

Square

Snake

back: 6

self new

© Oscar Nierstrasz
About the Buttons
**Metaclasses in 7 points**

1. Every object is an instance of a class
2. Every class eventually inherits from Object
3. Every class is an instance of a metaclass
4. The metaclass hierarchy parallels the class hierarchy
5. **Every metaclass inherits from Class and Behavior**
6. Every metaclass is an instance of Metaclass
7. The metaclass of Metaclass is an instance of Metaclass
5. Every metaclass inherits from Class and Behavior

Every class is-a Class =
— The metaclass of every class inherits from Class
Where is `new` defined?
Responsibilities of Behavior

> **Behavior**

  - Minimum state necessary for objects that have instances.
  - Basic interface to the compiler.
  - State:
    - *class hierarchy link, method dictionary, description of instances (representation and number)*
  - Methods:
    - *creating a method dictionary, compiling method*
    - *instance creation (new, basicNew, new:, basicNew:)*
    - *class hierarchy manipulation (superclass:, addSubclass:)*
    - *accessing (selectors, allSelectors, compiledMethodAt: )*
    - *accessing instances and variables (allInstances, instVarNames)*
    - *accessing class hierarchy (superclass, subclasses)*
    - *testing (hasMethods, includesSelector, canUnderstand:, inheritsFrom:, isVariable)*
Responsibilities of ClassDescription

> **ClassDescription**

  - adds a number of facilities to basic Behavior:
    - named instance variables
    - category organization for methods
    - the notion of a name (abstract)
    - maintenance of Change sets and logging changes
    - most of the mechanisms needed for fileOut

  - **ClassDescription** is an abstract class: its facilities are intended for inheritance by the two subclasses, **Class** and **Metaclass**.
Responsibilities of Class

> **Class**
  - represents the common behavior of all classes
    - *name, compilation, method storing, instance variables* ...
  - representation for classVariable names and shared pool variables (addClassVarName:, addSharedPool:, initialize)
  - **Class inherits from Object because Class is an Object**
    - *Class knows how to create instances, so all metaclasses should inherit ultimately from Class*
Metaclasses in 7 points

1. Every object is an instance of a class
2. Every class eventually inherits from Object
3. Every class is an instance of a metaclass
4. The metaclass hierarchy parallels the class hierarchy
5. Every metaclass inherits from Class and Behavior
6. **Every metaclass is an instance of Metaclass**
7. The metaclass of Metaclass is an instance of Metaclass
6. Every metaclass is an instance of Metaclass
Metaclass Responsibilities

> Metaclass
  - Represents common metaclass Behavior
    - instance creation (subclassOf:)
    - creating initialized instances of the metaclass’s sole instance
    - initialization of class variables
    - metaclass instance protocol (name:inEnvironment:subclassOf:....)
    - method compilation (different semantics can be introduced)
    - class information (inheritance link, instance variable, ...)
Metaclasses in 7 points

1. Every object is an instance of a class
2. Every class eventually inherits from Object
3. Every class is an instance of a metaclass
4. The metaclass hierarchy parallels the class hierarchy
5. Every metaclass inherits from Class and Behavior
6. Every metaclass is an instance of Metaclass
7. The metaclass of Metaclass is an instance of Metaclass
7. The metaclass of Metaclass is an instance of Metaclass
Navigating the metaclass hierarchy

```smalltalk
MetaclassHierarchyTest>>testHierarchy

"The class hierarchy"
self assert: Snake name = 'Snake'.
self assert: Snake superclass name = 'Square'.
self assert: Snake superclass superclass name = 'Object'.

"The parallel metaclass hierarchy"
self assert: Snake class name = 'Snake class'.
self assert: Snake class superclass name = 'Square class'.
self assert: Snake class superclass superclass name = 'Object class'.
...
self assert: Snake class superclass superclass superclass superclass superclass name = 'Class'.
...
self assert: Snake class superclass superclass superclass superclass super class name = 'Object'.

"The Metaclass hierarchy"
self assert: Snake class class name = 'Metaclass'.
self assert: Snake class class class name = 'Metaclass class'.
self assert: Snake class class class class name = 'Metaclass'.
```
Roadmap

- Metaclasses in 7 points
- Indexed Classes
- Class Instance Variables
- Class Variables
- Pool Dictionaries
Two ways to represent objects

> Named or indexed instance variables
   > Named: name of GamePlayer
   > Indexed: #(Jack Jill) at: 1

> Or looking at them in another way:
   > Objects with pointers to other objects
   > Objects with arrays of bytes (word, long)
   > Difference for efficiency reasons:
     > arrays of bytes (like C strings) are faster than storing an array of pointers, each pointing to a single byte.
### Different methods to create classes

<table>
<thead>
<tr>
<th>Indexed</th>
<th>Named Definition Method</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Yes</td>
<td>#subclass: ...</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>#variableSubclass: ...</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>#variableByteSubclass: ...</td>
</tr>
</tbody>
</table>

> See the subclass creation protocol of Class

> **Constraints**

  > *Pointer classes* defined using `#subclass:` support any kind of subclasses

  > *Byte classes* defined using `#variableSubclass:` can only have: `variableSubclass:` or `variableByteSubclass:` subclasses
Testing methods

> See testing protocols of `Behavior`:
  – `isPointers`, `isBits`, `isBytes`, `isFixed`, `isVariable`
  – `kindOfSubclass`
Defining Indexed Classes

> Example — instantiating an Array:

```smalltalk
Array new: 4
#(nil nil nil nil)
```

```smalltalk
ArrayedCollection variableSubclass: #Array
instanceVariableNames: ''
classVariableNames: ''
poolDictionaries: ''
category: 'Collections-Arrayed'
```

```smalltalk
#(1 2 3 4) class isVariable
true
```
Defining an Indexed Class

Object

variableSubclass: #IndexedObject
instanceVariableNames: ''
classVariableNames: ''
poolDictionaries: ''
category: ''

(IndexedObject new: 2)
at: 1 put: 'Jack';
at: 2 put: 'Jill';
at: 1

'Jack'
Indexed Classes / Instance Variables

> An indexed variable is implicitly added to the list of instance variables
  > Only one indexed instance variable per class
  > Access with at: and at:put:
    > NB: answers the value, not the receiver

> Subclasses should also be indexed
Roadmap

- Metaclasses in 7 points
- Indexed Classes
- **Class Instance Variables**
- Class Variables
- Pool Dictionaries
Class Instance Variables

> Class are objects too
  – Instances of their metaclass
    – *Methods looked up in the method dictionary of their metaclass*
  – Can also define instance variables

> When a metaclass defines a new instance variable, then its instance (a Class) gets a new variable
  – I.e., in addition to subclass, superclasses, methodDict...

> Use class instance variables to represent the private state of the class
  – E.g., number of instances, superclass etc.
    – *Not to represent information shared by all instances!*

© Oscar Nierstrasz
Example: the Singleton Pattern

> A class having only one instance
  — We keep the unique instance created in an instance variable

WebServer class
  instanceVariableNames: 'uniqueInstance'

WebServer class>>new
  self error: 'Use uniqueInstance to get the unique instance'

WebServer class>>uniqueInstance
  uniqueInstance isNil
    ifTrue: [uniqueInstance := self basicNew initialize].
  ^ uniqueInstance
Roadmap

- Metaclasses in 7 points
- Indexed Classes
- Class Instance Variables
- **Class Variables**
- Pool Dictionaries
To share information amongst all instances of a class, use a “class variable”

- Shared and directly accessible by all the instances of the class and subclasses
- Accessible to both instance and class methods
- Begins with an uppercase letter
Class variables should be initialized by an initialize method on the class side.

Magnitude subclass: #Date
  instanceVariableNames: 'julianDayNumber'
  classVariableNames: 'DaysInMonth FirstDayOfMonth
                      MonthNames SecondsInDay WeekDayNames'
  poolDictionaries: ''
  category: 'Kernel-Magnitudes'

Date class>>initialize
  ...
  WeekDayNames := #(Sunday Monday ...).
  MonthNames := #(January February ... ).
  DaysInMonth := #(31 28 31 30 31 30 31 31 30 31 30 31).
ClassVariables vs. Instance Variables

@@ This is not like that anymore in Squeak Sniffffff@@@ but I liked the example
Roadmap

- Metaclasses in 7 points
- Indexed Classes
- Class Instance Variables
- Class Variables
- Pool Dictionaries
Pool Dictionaries

> A Pool Dictionary is a shared variable
  — Begins with an uppercase letter.
  — Shared by a group of classes not linked by inheritance.

> Each class possesses its own pool dictionary (containing pool variables).
  — They are not inherited.

> *Don’t use them!*
Examples of Pool Dictionaries

ArrayedCollection subclass: #Text
  instanceVariableNames: 'string runs'
  classVariableNames: ''
  poolDictionaries: 'TextConstants'
  category: 'Collections-Text'

> Elements stored into TextConstants like Ctrl, CR, ESC, Space can be directly accessed from all the classes like ParagraphEditor....

> **Hint:** You can inspect any Pool Dictionary
Pool Dictionaries are stored in the Smalltalk system dictionary

(Smalltalk at: #TextConstants) at: #ESC
Accessing globals

> Use message-sending instead of directly accessing pool variables

```
stream nextPut: Lf "A pool variable visible to the class"
```

VS.

```
stream nextPut: Character lf
```
What you should know!

- What does is-a mean?
- What is the difference between sending a message to an object and to its class?
- What are the responsibilities of a metaclass?
- What is the superclass of Object class?
- Where is new defined?
- What is the difference between class variables and class instance variables?
Can you answer these questions?

- Why are there no explicit metaclasses?
- When should you override `new`?
- Why don’t metaclasses inherit from `Class`?
- Are there any classes that don’t inherit from `Object`?
- Is `Metaclass` a `Class`? Why or why not?
- Where are the methods `class` and `superclass` defined?
- When should you define an indexed class?
- Are Java static variables just like class variables or class instance variables?
- Where is the `SystemDictionary` Smalltalk defined?
License

> http://creativecommons.org/licenses/by-sa/2.5/

You are free:
• to copy, distribute, display, and perform the work
• to make derivative works
• to make commercial use of the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor.

Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

• For any reuse or distribution, you must make clear to others the license terms of this work.
• Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.