

9. Understanding Classes and Metaclasses



Birds-eye view



Reify your metamodel — A fully reflective system models its own metamodel.



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

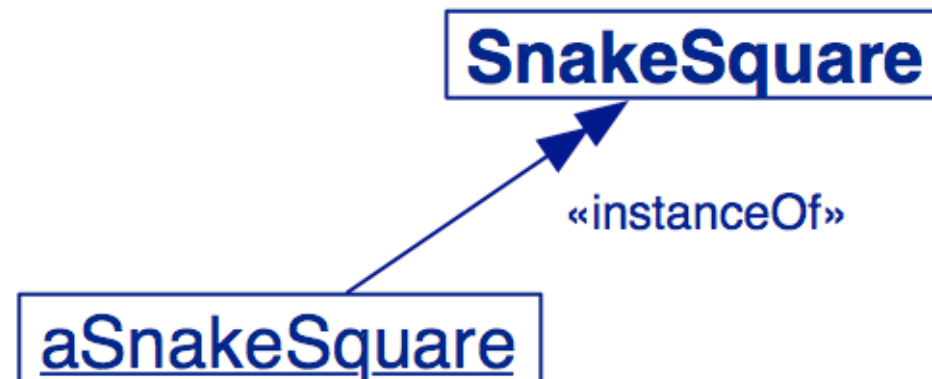
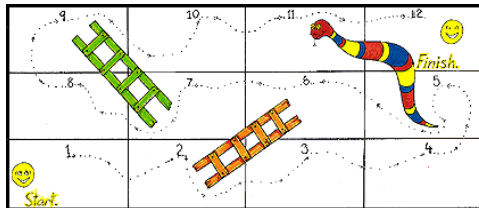
1. Every object is an instance of a class
2. Every class eventually inherits from Object
3. Every class is an instance of a metaclass
4. The metaclass hierarchy parallels the class hierarchy
5. Every metaclass inherits from Class and Behavior
6. Every metaclass is an instance of Metaclass
7. The metaclass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

Adapted from Goldberg & Robson, *Smalltalk-80 — The Language*

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

1. Every object is an instance of a class



Remember the Snakes and Ladders Board Game ...

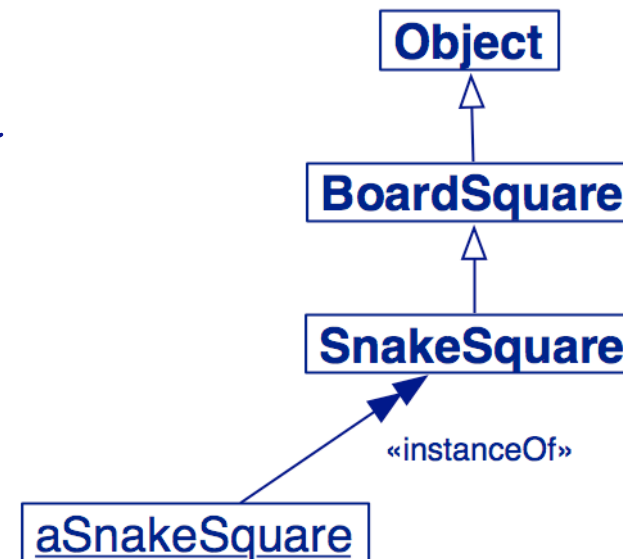
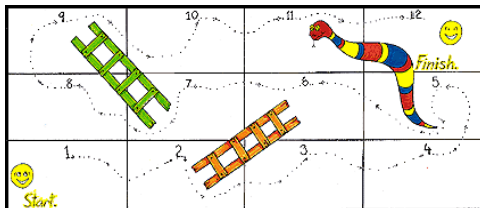
Metaclasses in 7 points

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

2. Every class inherits from Object

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

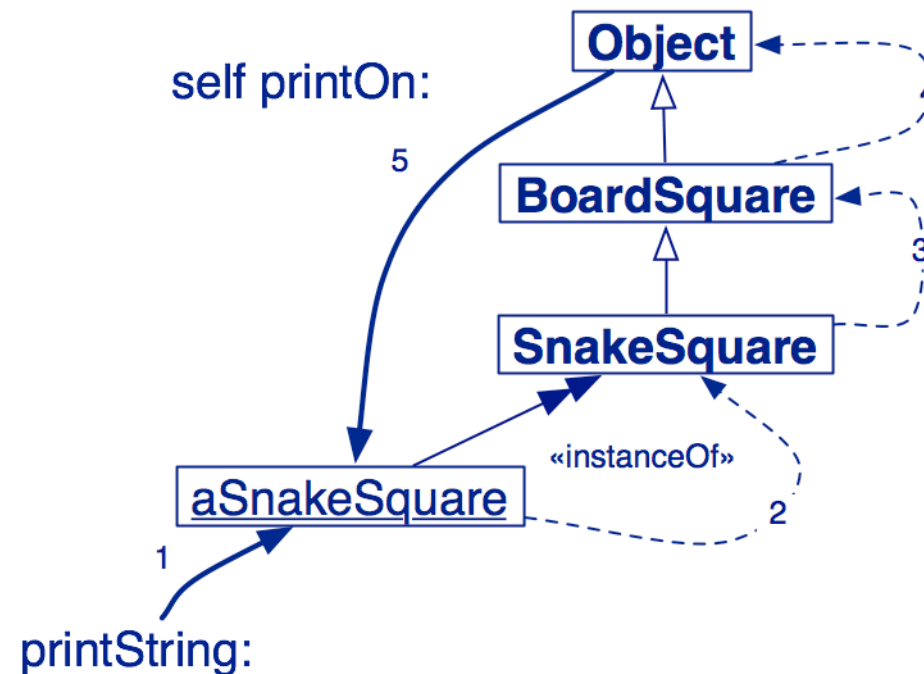
aSnakeSquare is-a SnakeSquare
and is-a BoardSquare
and is-an Object



Caveat: in Pharo, 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



Responsibilities of Object

> **Object**

- represents the common object behavior
 - *error-handling, halting ...*
- all classes should inherit ultimately from Object

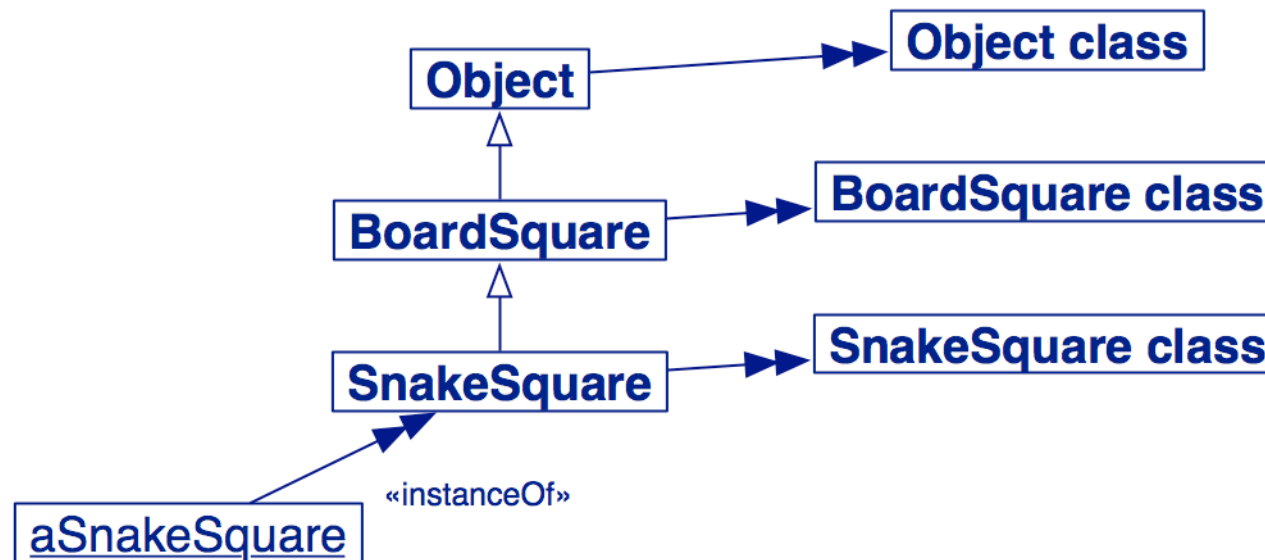
Metaclasses in 7 points

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

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`



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

```
BoardSquare allSubclasses  
SnakeSquare allSubclasses
```

```
SnakeSquare allInstances  
SnakeSquare instVarNames
```

```
SnakeSquare back: 5
```

```
SnakeSquare selectors
```

```
SnakeSquare canUnderstand: #new  
SnakeSquare canUnderstand: #setBack:
```

```
a Set(SnakeSquare FirstSquare LadderSquare)  
a Set()
```

```
an Array(<-2[6] <-4[11] <-6[11])  
#('back')
```

```
<-5[nil]
```

```
an IdentitySet(#setBack:  
                #printOn: #destination)
```

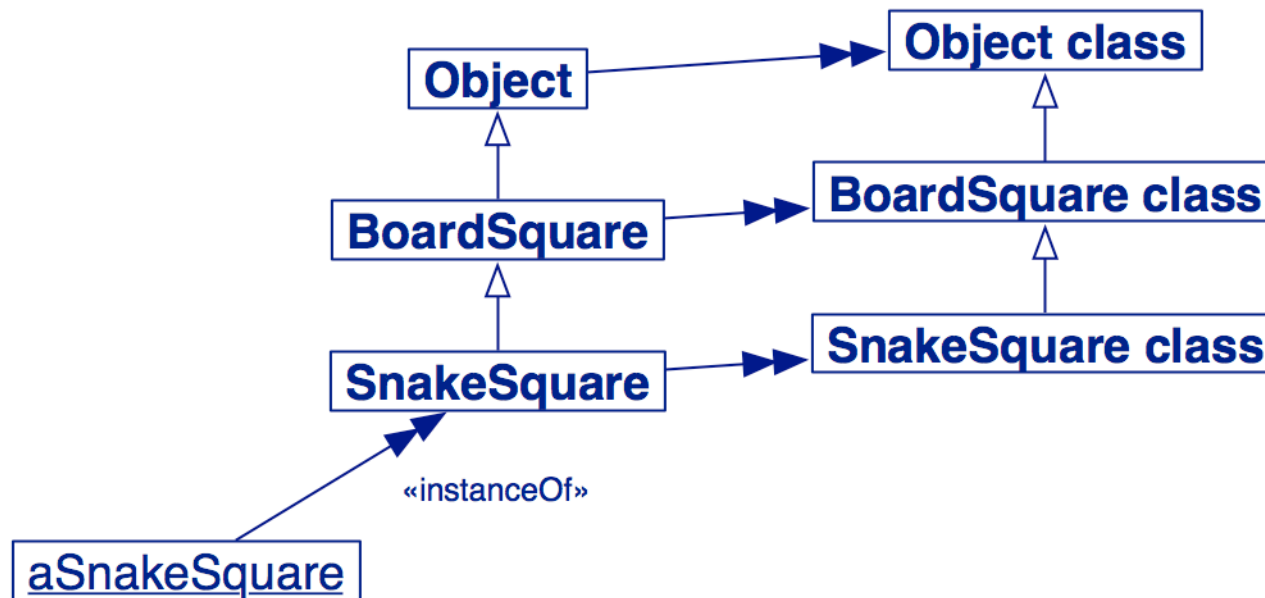
```
false
```

```
true
```

Metaclasses in 7 points

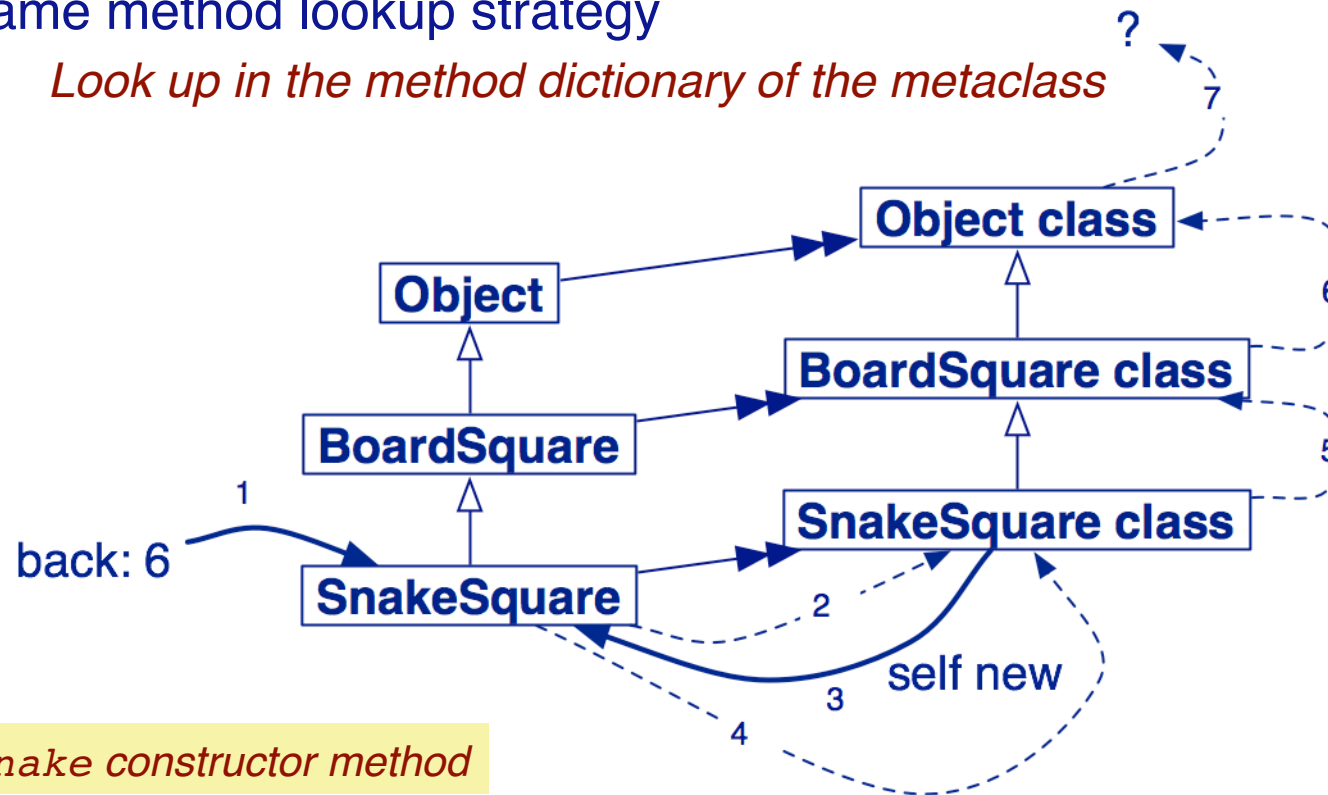
1. Every object is an instance of a class
2. Every class eventually inherits from Object
3. Every class is an instance of a metaclass
4. **The metaclass hierarchy parallels the class hierarchy**
5. Every metaclass inherits from Class and Behavior
6. Every metaclass is an instance of Metaclass
7. The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

4. The metaclass hierarchy parallels the class hierarchy



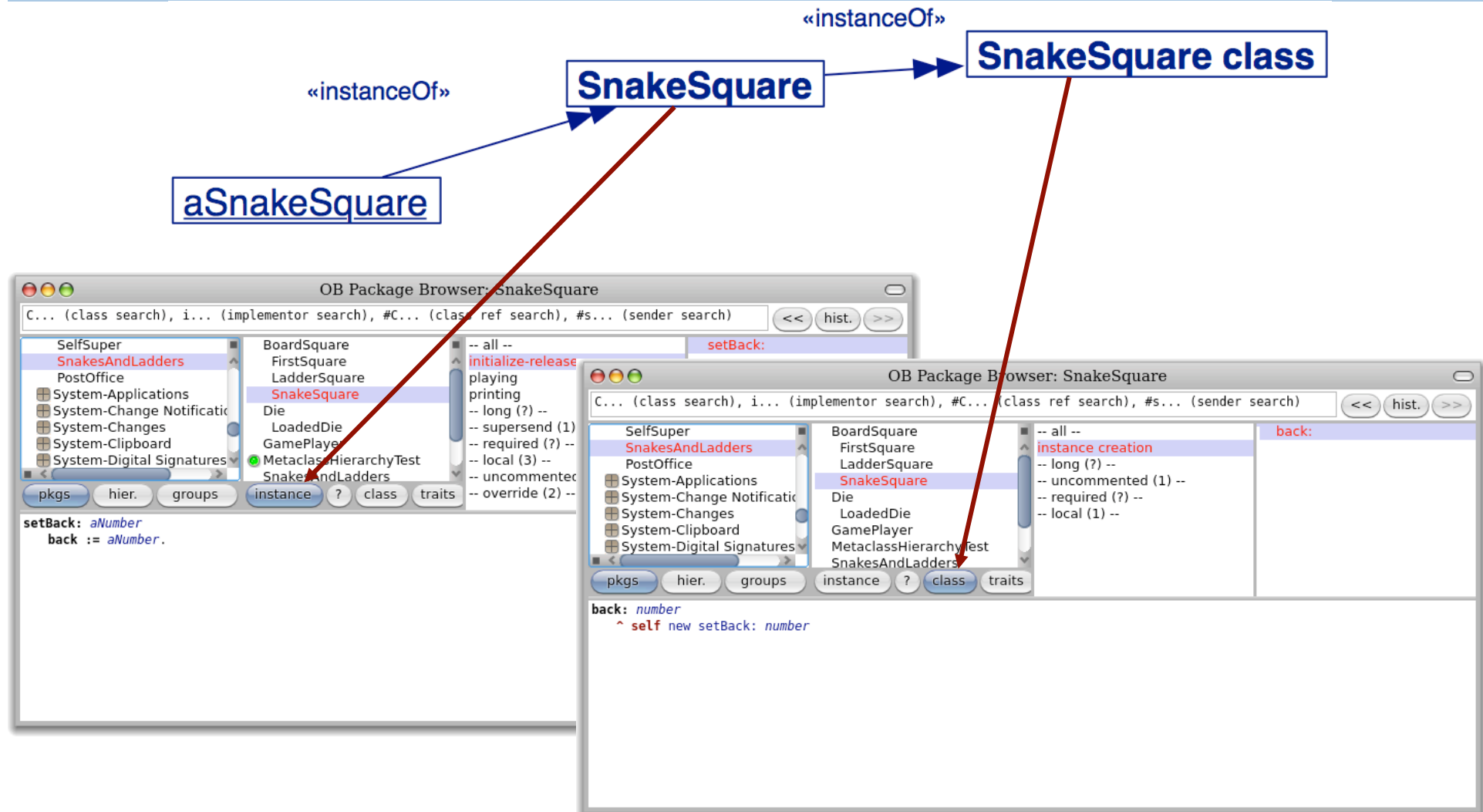
Uniformity between Classes and Objects

- > 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*



back: is a Snake constructor method

About the Buttons



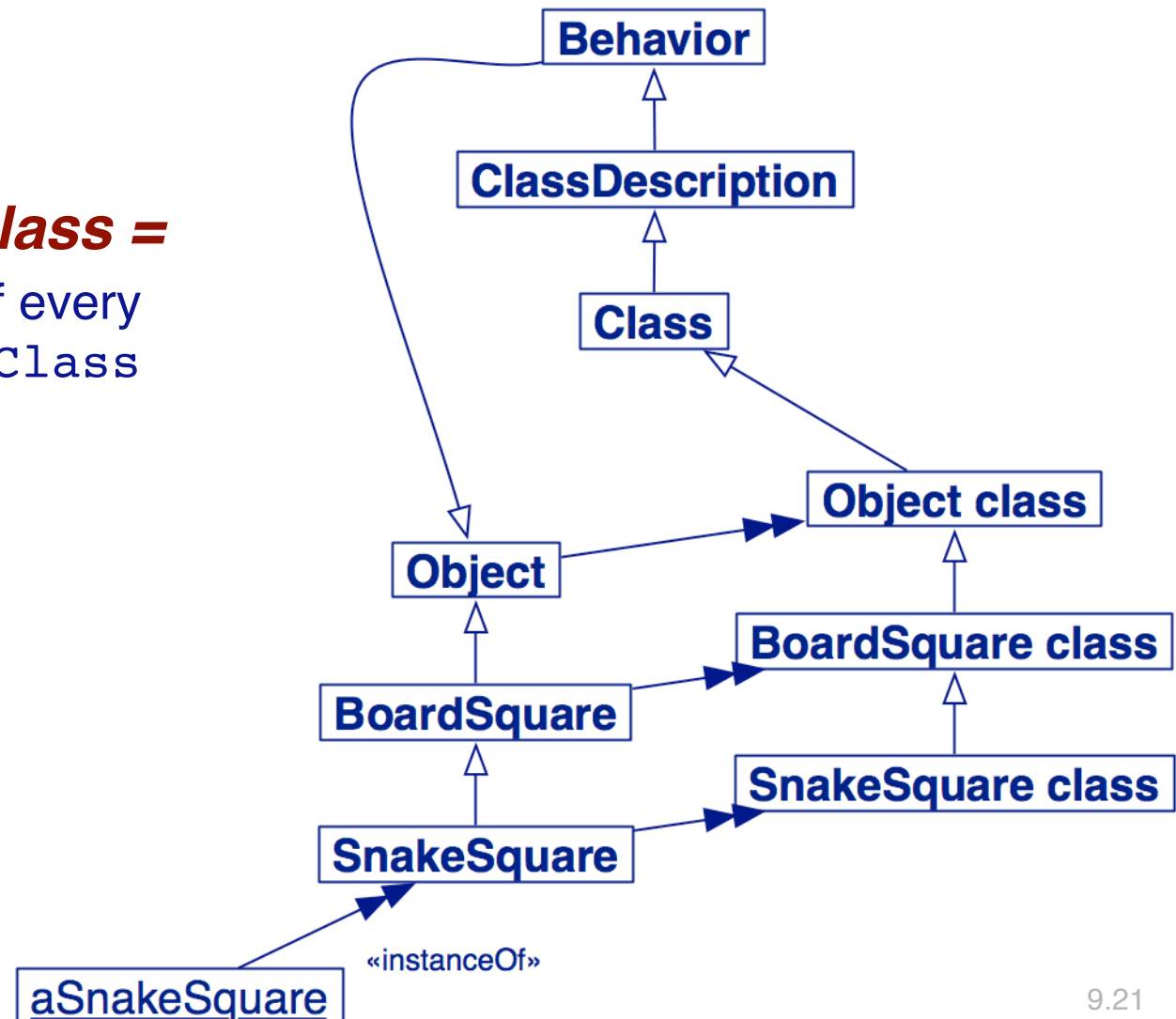
Metaclasses in 7 points

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

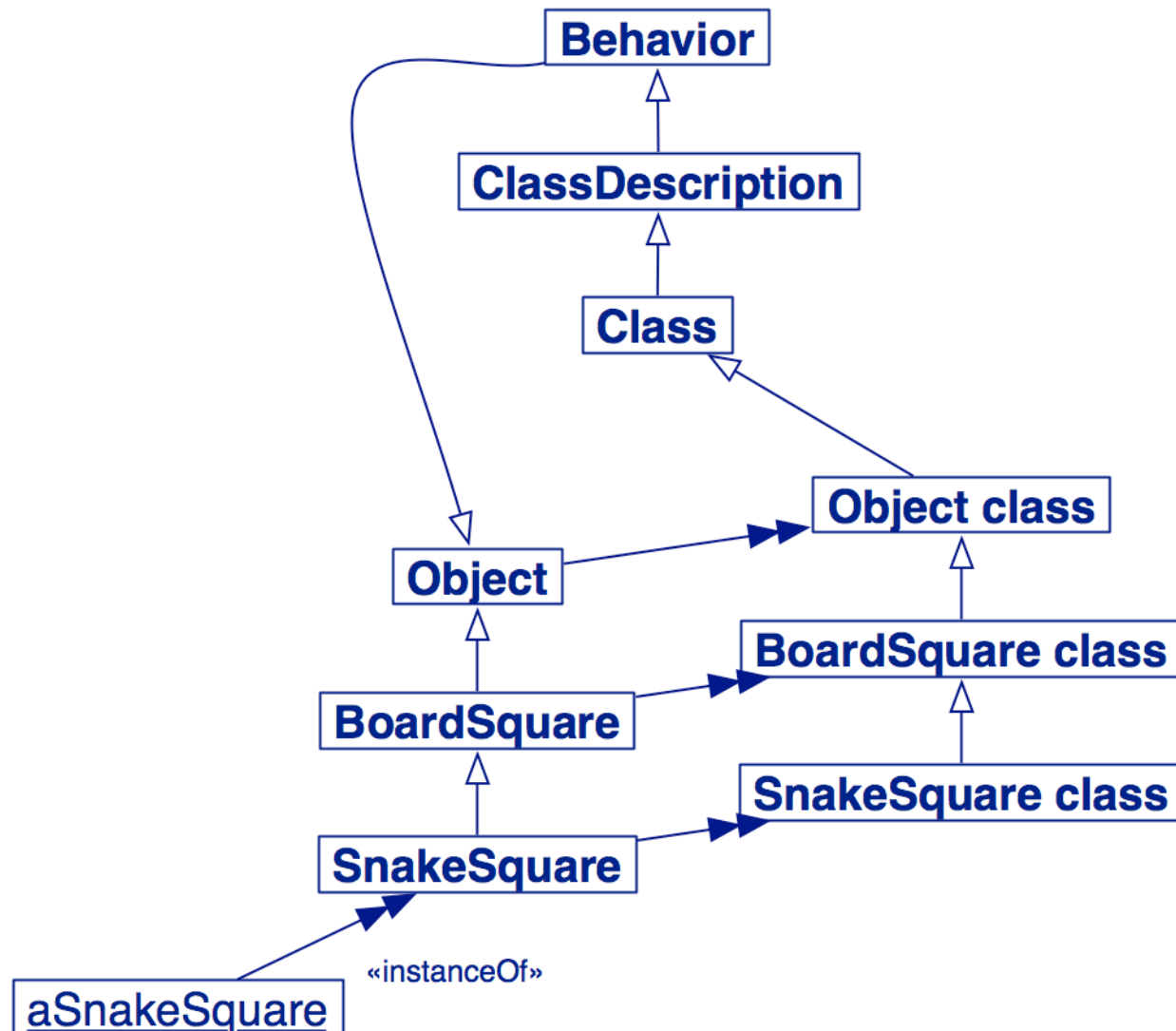
5. Every metaclass inherits from Class and Behavior

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

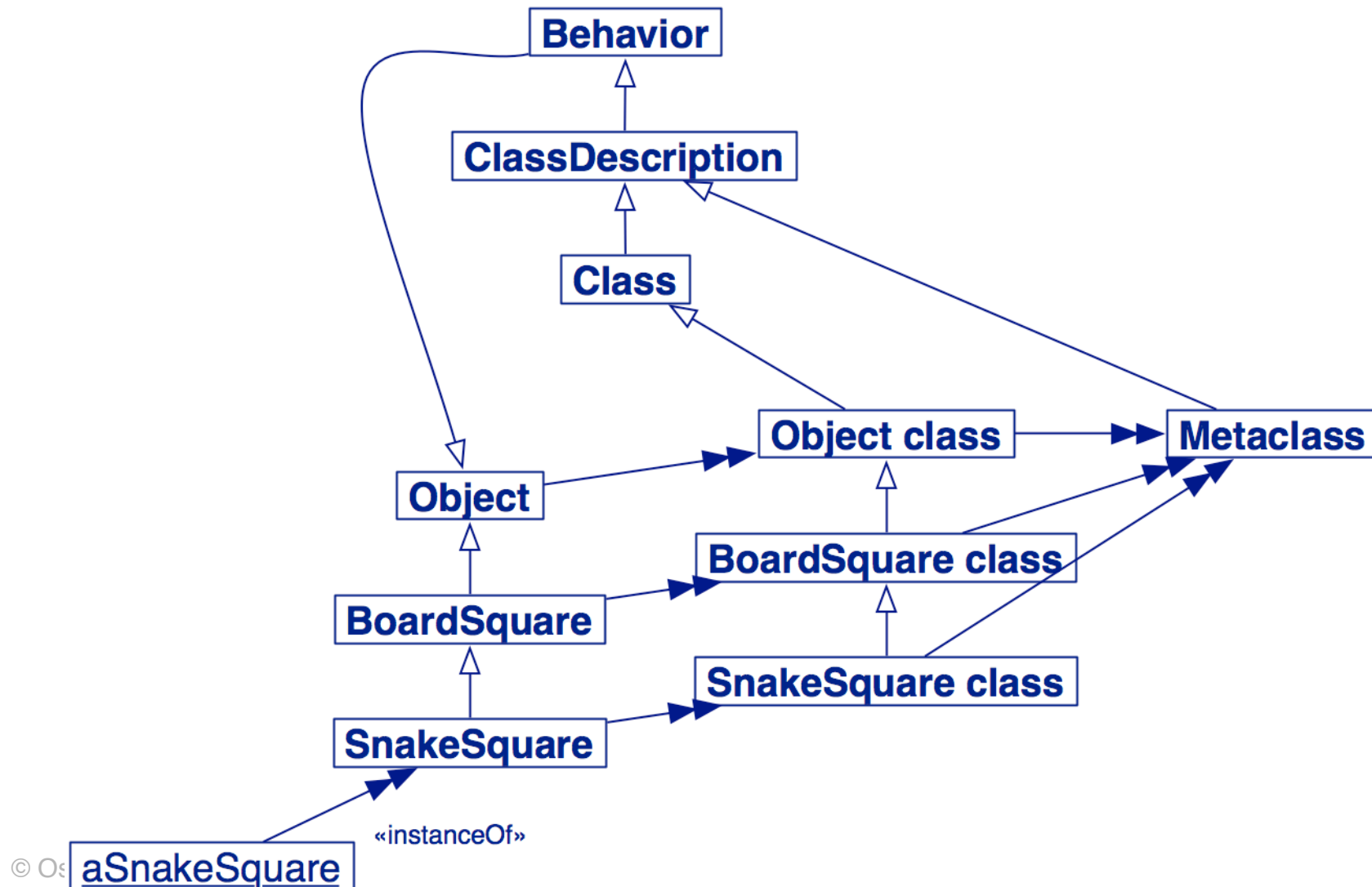
> **Class**

- 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

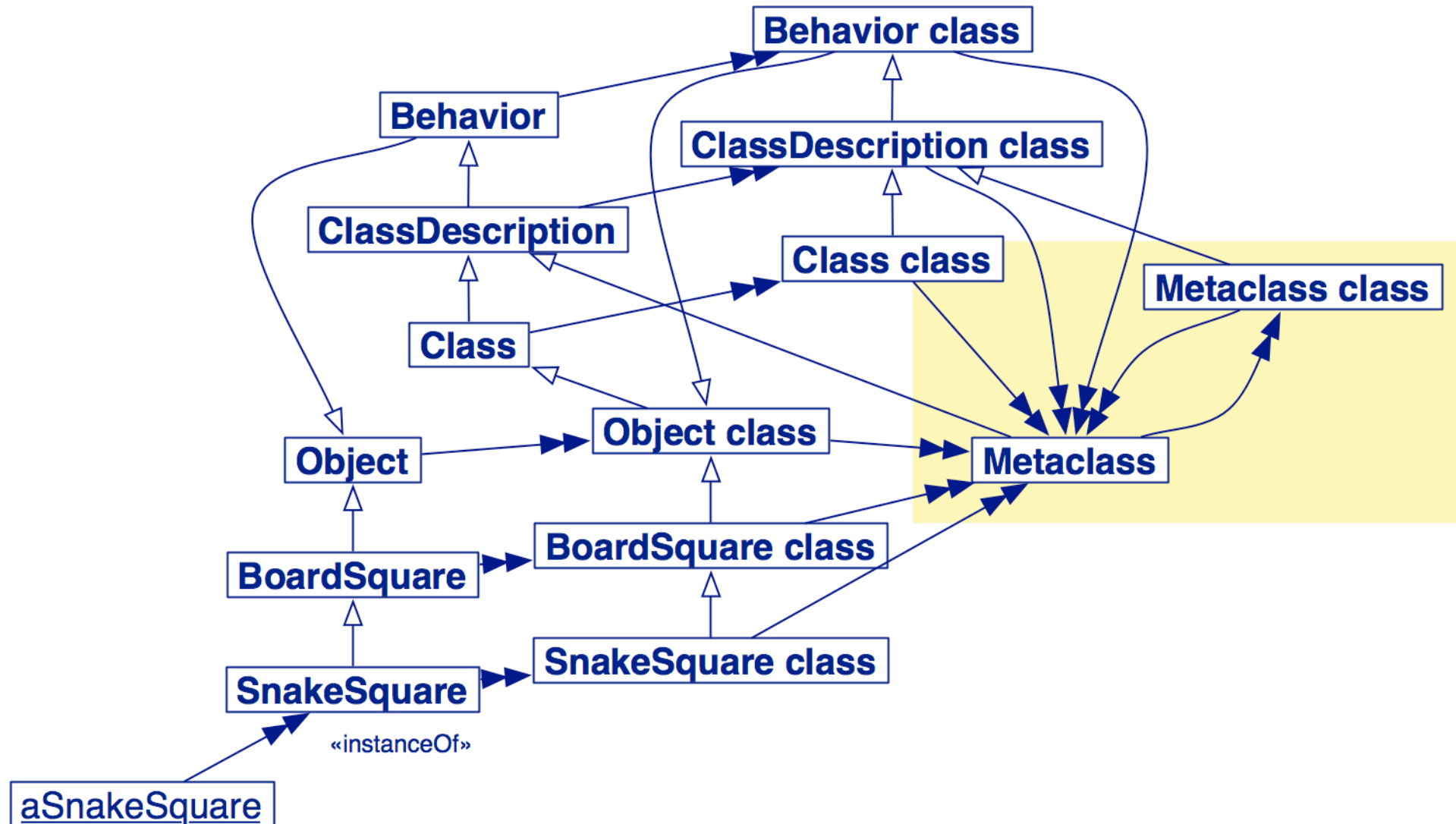
— Represents common metaclass Behavior

- *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

```
MetaclassHierarchyTest>>testHierarchy
  "The class hierarchy"
  self assert: SnakeSquare superclass = BoardSquare.
  self assert: BoardSquare superclass = Object.
  self assert: Object superclass superclass = nil.
  "The parallel metaclass hierarchy"
  self assert: SnakeSquare class name = 'SnakeSquare class'.
  self assert: SnakeSquare class superclass = BoardSquare class.
  self assert: BoardSquare class superclass = Object class.
  self assert: Object class superclass superclass = Class.
  self assert: Class superclass = ClassDescription.
  self assert: ClassDescription superclass = Behavior.
  self assert: Behavior superclass = Object.
  "The Metaclass hierarchy"
  self assert: SnakeSquare class class = Metaclass.
  self assert: BoardSquare class class = Metaclass.
  self assert: Object class class = Metaclass.
  self assert: Class class class = Metaclass.
  self assert: ClassDescription class class = Metaclass.
  self assert: Behavior class class = Metaclass.
  self assert: Metaclass superclass = ClassDescription.
  "The fixpoint"
  self assert: Metaclass class class = 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

Indexed	Named Instance Variables	Definition Method
No	Yes	<code>#subclass: ...</code>
Yes	Yes	<code>#variableSubclass: ...</code>
Yes	No	<code>#variableByteSubclass: ...</code>

- > 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

```
Object allSubclasses select: [:class | class isBytes]
```

```
a Set(ByteArray MwSynchronizationWrapper  
MwBlockMethodWrapper ExternalAddress MCMockClassH  
LargeNegativeInteger LargePositiveInteger  
ByteSymbol MwCountMethodWrapper MwTimeMethodWrapper  
MwMethodWrapper MwBlockHandlerMethodWrapper  
ByteString MwCalledMethodWrapper UUID  
CompiledMethod)
```

Defining Indexed Classes

> Example — instantiating an Array:

```
Array new: 4
```

```
 #(nil nil nil nil)
```

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

```
 #(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!*

Example: the Singleton Pattern

- > A class with 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



Class Variable = Shared Variable

- > 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

Initializing class variables

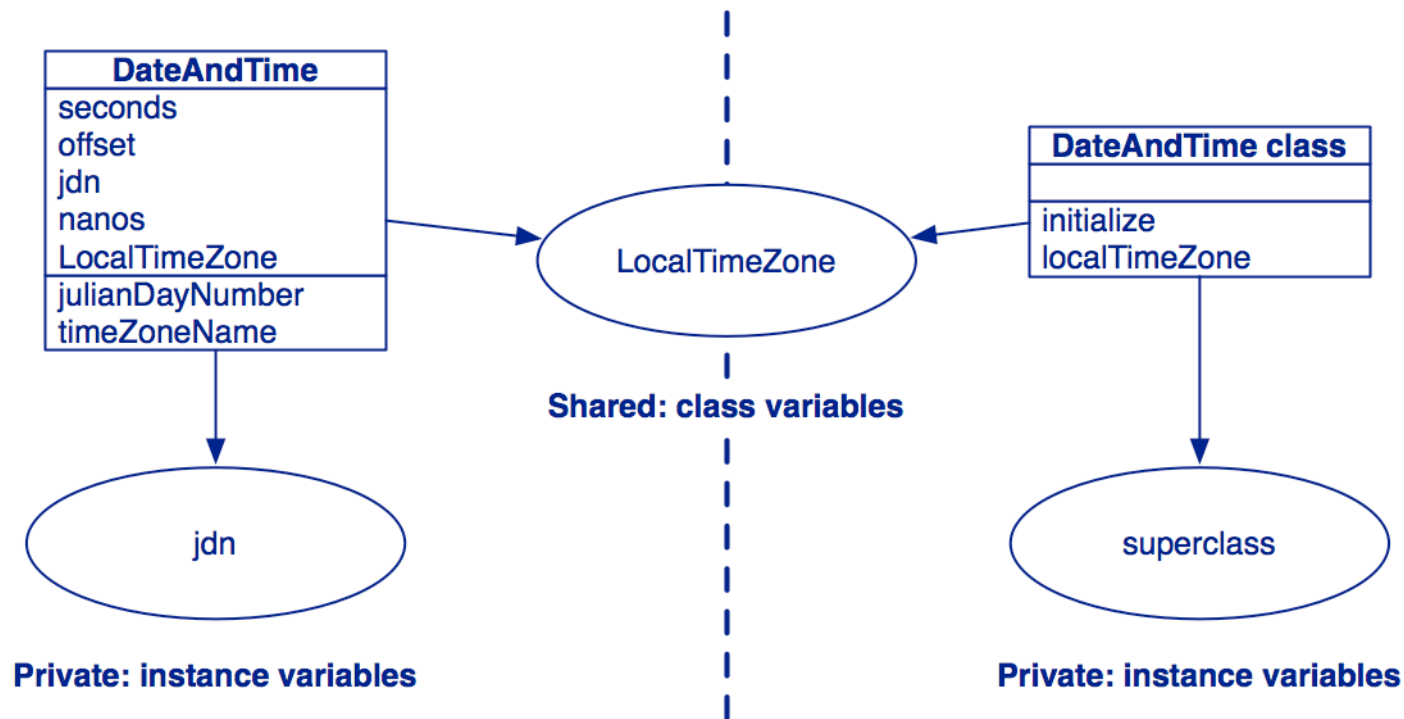
- > Class variables should be initialized by an `initialize` method on the class side, or by lazy initialization

```
Magnitude subclass: #DateAndTime
  instanceVariableNames: 'seconds offset jdn nanos'
  classVariableNames: 'LocalTimeZone'
  poolDictionaries: 'ChronologyConstants'
  category: 'Kernel-Chronology'

Date class>>localTimeZone
  "Answer the local time zone"

  ^ LocalTimeZone ifNil: [ LocalTimeZone := TimeZone default ]
```

ClassVariables vs. Instance Variables



Roadmap

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



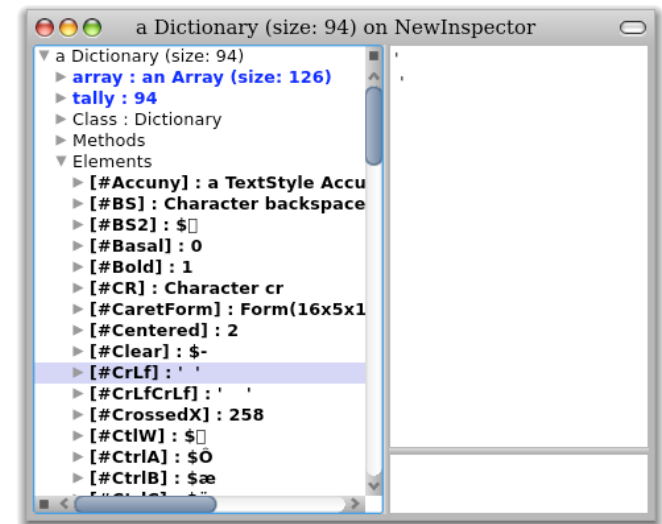
Pool Dictionaries

- > A Pool Dictionary is a shared variable
 - Begins with a 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'
```

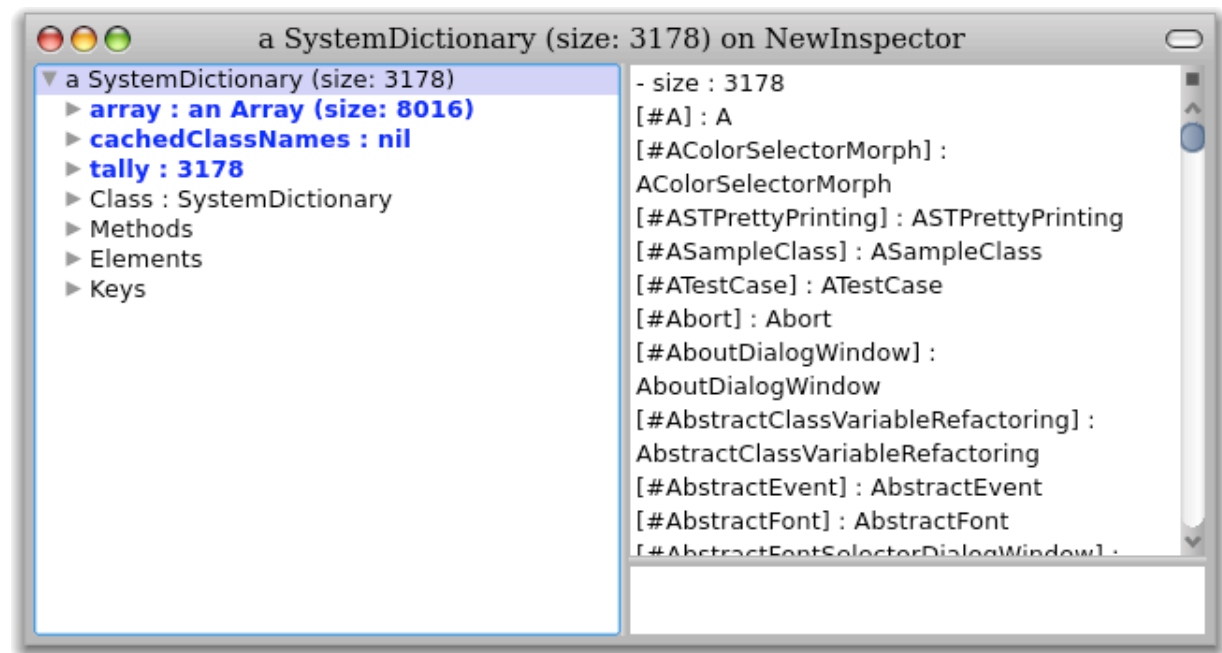
- > Elements stored into TextConstants like Ctrl, CR, ESC, Space can be directly accessed from all the classes like ParagraphEditor....
- > **Hint:** You can inspect any Pool Dictionary



Smalltalk System Dictionary

- > Pool Dictionaries are stored in the Smalltalk system dictionary

Smalltalk inspect



(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/3.0/>



Attribution-ShareAlike 3.0 Unported

You are free:

- to Share** — to copy, distribute and transmit the work
- to Remix** — to adapt the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page.

Any of the above conditions can be waived if you get permission from the copyright holder.

Nothing in this license impairs or restricts the author's moral rights.