Some Advanced Points on Classes

Stéphane Ducasse
Stephane.Ducasse@univ-savoie.fr
http://www.listic.univ-savoie.fr/~ducasse/

License: CC-Attribution-ShareAlike 2.0

http://creativecommons.org/licenses/by-sa/2.0/

creative commons

COMMONS DEED

Attribution-ShareAlike 2.0

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 give the original author credit.

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.

This is a human-readable summary of the Legal Code (the full license).

Instantiation

- ***Basic class instantiation***

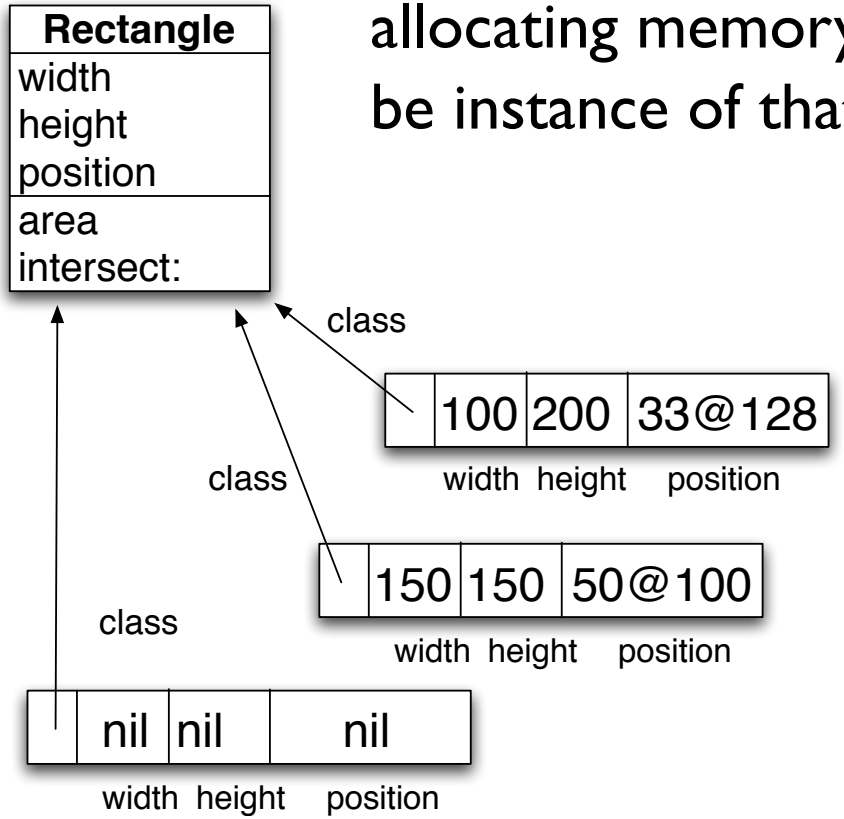
Object Instantiation

Objects can be created by:

- Direct Instance creation: new/new:
- Messages to instances that create other objects
- Class specific instantiation messages

Object Creation

- When a class creates an object =
  allocating memory + marking it to be instance of that class



Instance Creation with new

aClass new

returns a newly and ***UNINITIALIZED*** instance

```
OrderedCollection new -> OrderedCollection ()
Packet new -> aPacket
```

Default instance variable values are nil
nil is an instance of UndefinedObject and only understands a limited set of messages

Messages to Instances

Messages to Instances that create Objects

1 to: 6	(an interval)
1@2	(a point)
(0@0) extent: (100@100)	(a rectangle)
#lulu asString	(a string)
1 printString	(a string)
3 asFloat	(a float)
#(23 2 3 4) asSortedCollection	(a sortedCollection)

Opening the Box

1 to: 6
creates an interval

```
Number>>to: stop
   "Answer an Interval from the receiver up to the argument,
   stop, with each next element computed by incrementing the
   previous one by 1."

     ^Interval from: self to: stop by: 1
```

Strings...

I printString

```
Object>>printString
   "Answer a String whose characters are a description
   of the receiver."

    | aStream |
    aStream := WriteStream on: (String new: 16).
    self printOn: aStream.
    ^ aStream contents
```


Instance Creation

1@2
creates a point

```
Number>>@ y
   "Answer a new Point whose x value is the receiver and
   whose y value is the argument."

   <primitive: 18>
   ^ Point x: self y: y
```

Class-specific Messages

Array **with**: 1 **with**: 'lulu'

OrderedCollection **with**: 1 **with**: 2 **with**: 3

Rectangle **fromUser** -> 179@95 corner: 409@219

Browser **browseAllImplementorsOf**: #at:put:

Packet **send**: 'Hello mac' **to**: #mac

Workstation **withName**: #mac

new and new:

- **new:**/basicNew: is used to specify the size of the created instance

Array **new:** 4 -> #(nil nil nil nil)

- new/new: can be specialized to define customized creation
- basicNew/basicNew: should never be overridden
- #new/basicNew and new:/basicNew: are class methods

Outline

- Indexed Classes
- Classes as Objects
- Class Instance Variables and Methods
- Class Variables

Variable size instance

How do we represent objects whose size is variable such an array

Array new: 10

Array new: 15

Two Views on Classes

Named or ***indexed*** instance variables

Named: 'addressee' of Packet

Indexed: Array

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.

Types of Classes

Indexed	Named	Definition Method	Examples
No	Yes	#subclass:...	Packet
Yes	Yes	#variableSubclass:	Array
Yes	No	#variableByteSubclass	String

Method related to class types: #isPointers, #isBits, #isBytes, #isFixed, #isVariable, #kindOfSubclass

Constraints

Classes defined using #subclass: support any kind of subclasses

Classes defined using #variableSubclass: can only have: variableSubclass: or variableByteSubclass: subclasses

pointer classes and byte classes don't mix: e.g. only byte subclasses of byte classes.

Indexed Classes

For classes that need a variable number of instance variables

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

    Array new: 4 -> #(nil nil nil nil)
    #(1 2 3 4) class isVariable -> true
```

Indexed Classes

Indexed variable is implicitly added to the list of instance variables

Only one indexed instance variable per class

Access with #at: and #at:put:

(#at:put: answers the value, not the receiver)

Subclasses should also be indexed

Index access

First access: anInstance at: 1

#size returns the number of indexed instance variables

Instantiated with #new: max

```
|t|
t := (Array new: 4).
t at: 2 put: 'lulu'.
t at: 1 -> nil
```


Roadmap

- Indexed Classes
- Classes as Objects
- Class Instance Variables and Methods
- Class Variables

The Meaning of is-a

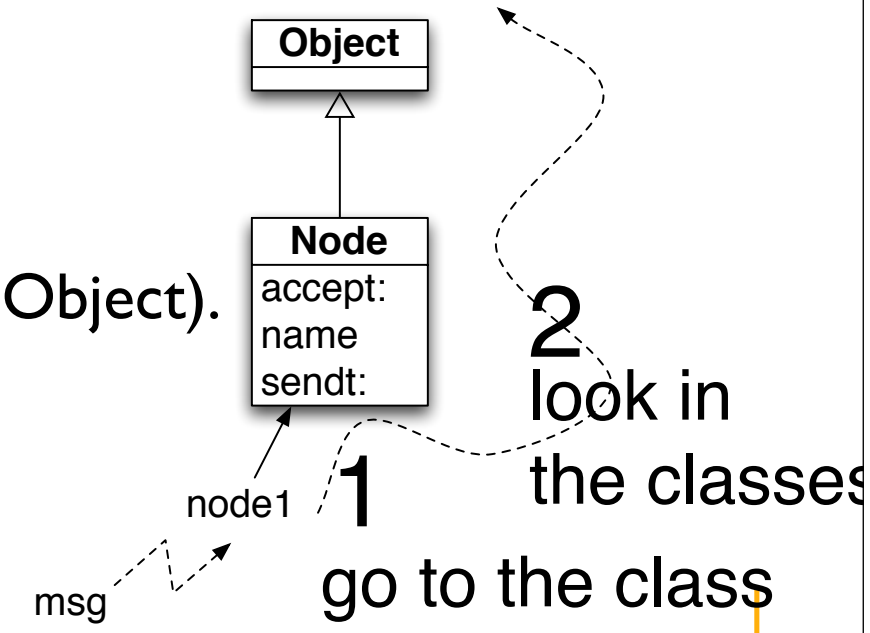
A class defines the structure and the behavior of all its instances.

Each instance possesses its own set of values.

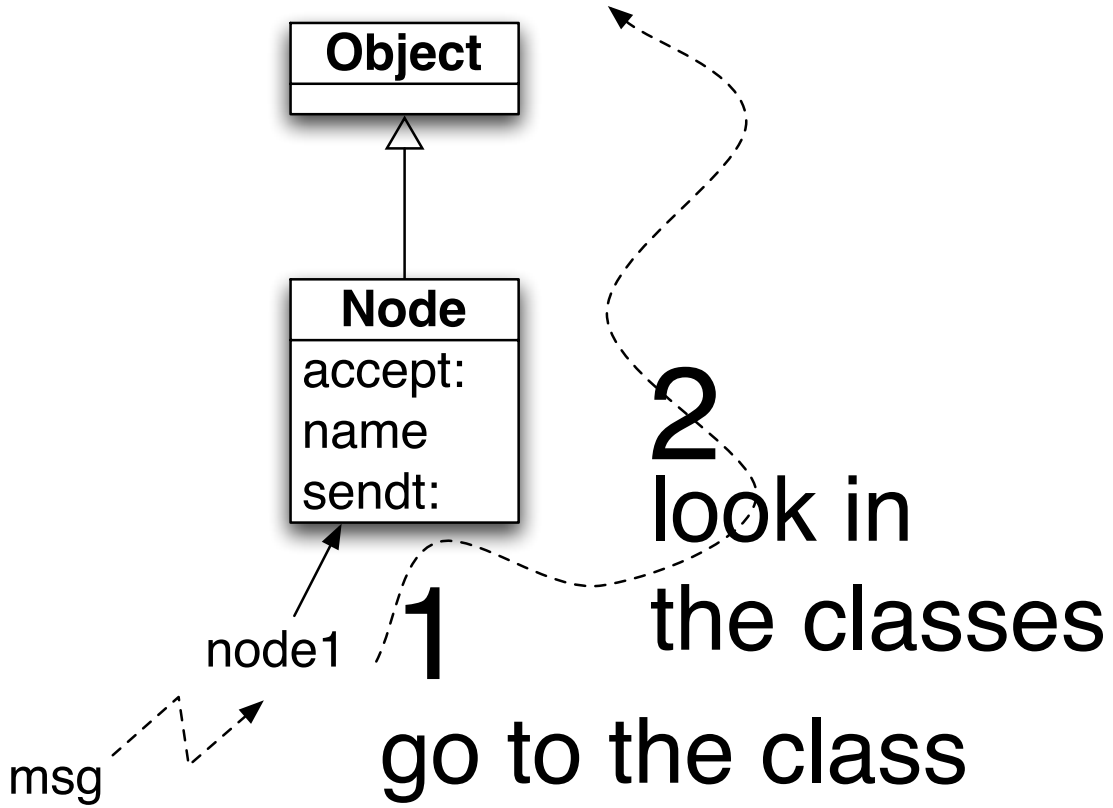
Instances share the behavior defined in their class with other instances via the instance of link.

The Meaning of Is-a

- Every object is an instance of a class.
- When anObject receives a message, the method is looked up in ***its class***
- And it continues possibly in its superclasses
- Every class is ultimately a subclass of Object (except Object).



Lookup...


Object
Node
accept:
name
sendt:
node1
msg
1
go to the class
2
look in
the classes

Remember: …

Example: macNode name
macNode is an instance of Workstation
=> name is looked up in the class Workstation
name is not defined in Workstation
=> lookup continues in Node
name is defined in Node
=> lookup stops + method executed

Roadmap

- Indexed Classes
- ***Classes as Objects***
- Class Instance Variables and Methods
- Class Variables

Classes and Objects

- Classes are objects too
- The same principle is true for objects and classes
- Same lookup strategy
- Everything that works at instance levels works at class level

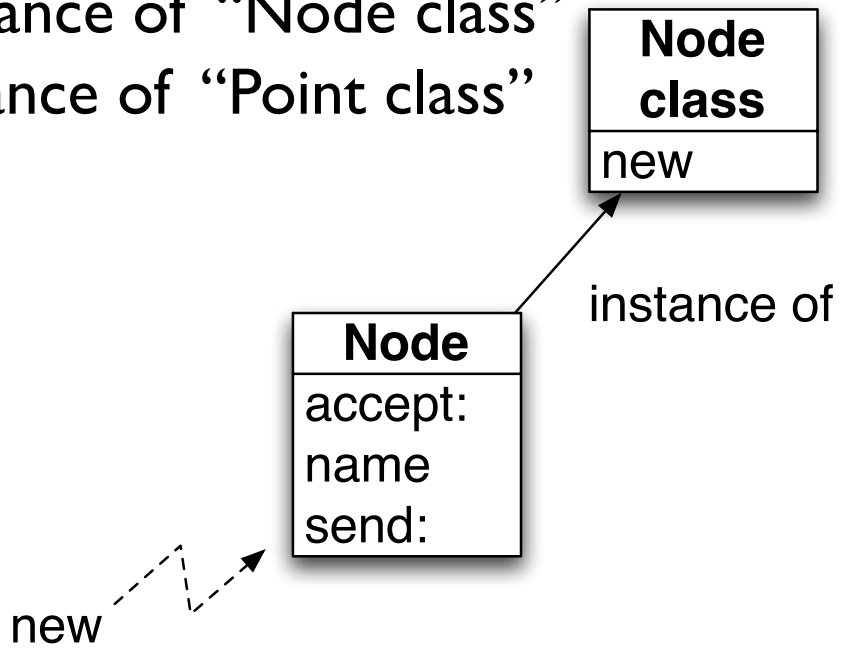
- In some language classes are not objects, still understanding it in Smalltalk will force you to really understand what instance/inheritance means

Class Responsibilities

- instance creation
- class information (inheritance link, instance variables, method compilation...)
- Examples:
  - Node allSubclasses -> OrderedCollection (WorkStation OutputServer Workstation File)
  - LanPrinter allInstances -> #()
  - Node instVarNames -> #('name' 'nextNode')
  - Workstation withName: #mac -> aWorkstation
  - Workstation selectors -> IdentitySet (#accept: #originate:)
  - Workstation canUnderstand: #nextNode -> true

A Class is an Object too…

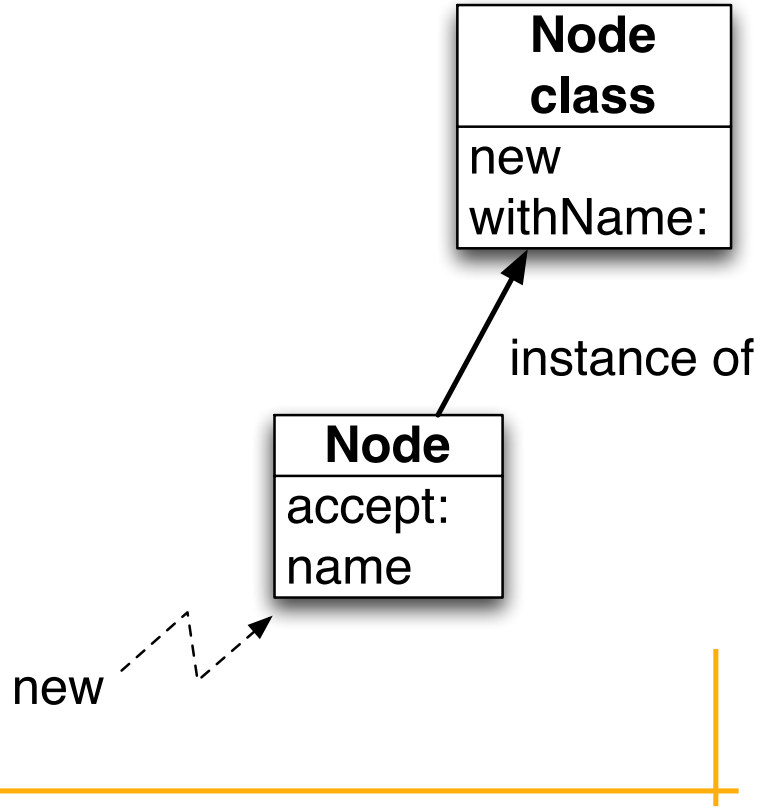
- Every class (X) is the unique instance of its associated metaclass named X class
- Example:
- Node is the unique instance of "Node class"
- Point is the unique instance of "Point class"



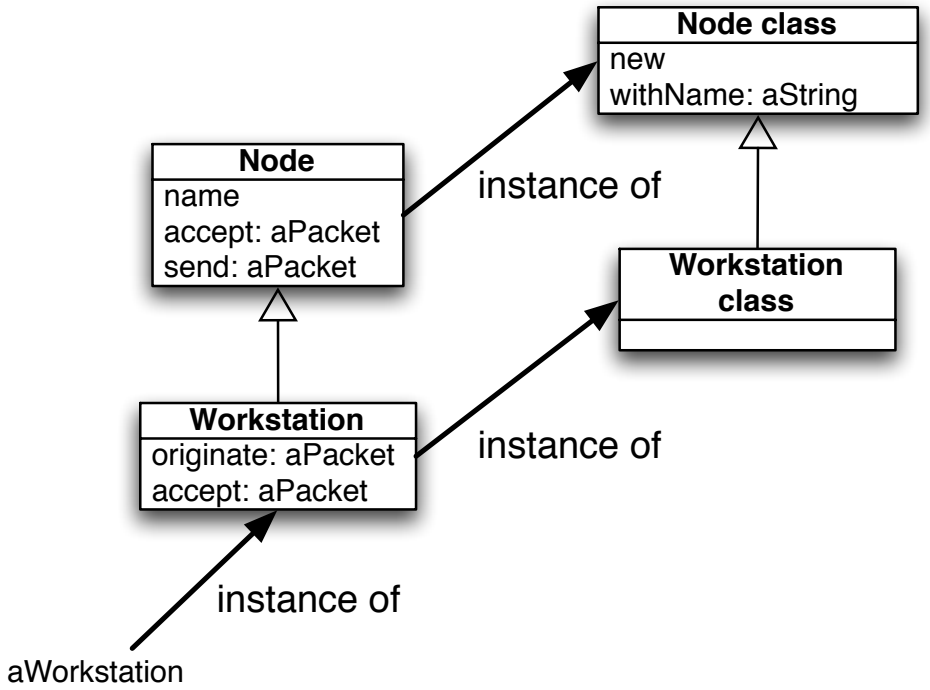
A Class is an Object too…

So messages sent to a class are looked up into the class of the class

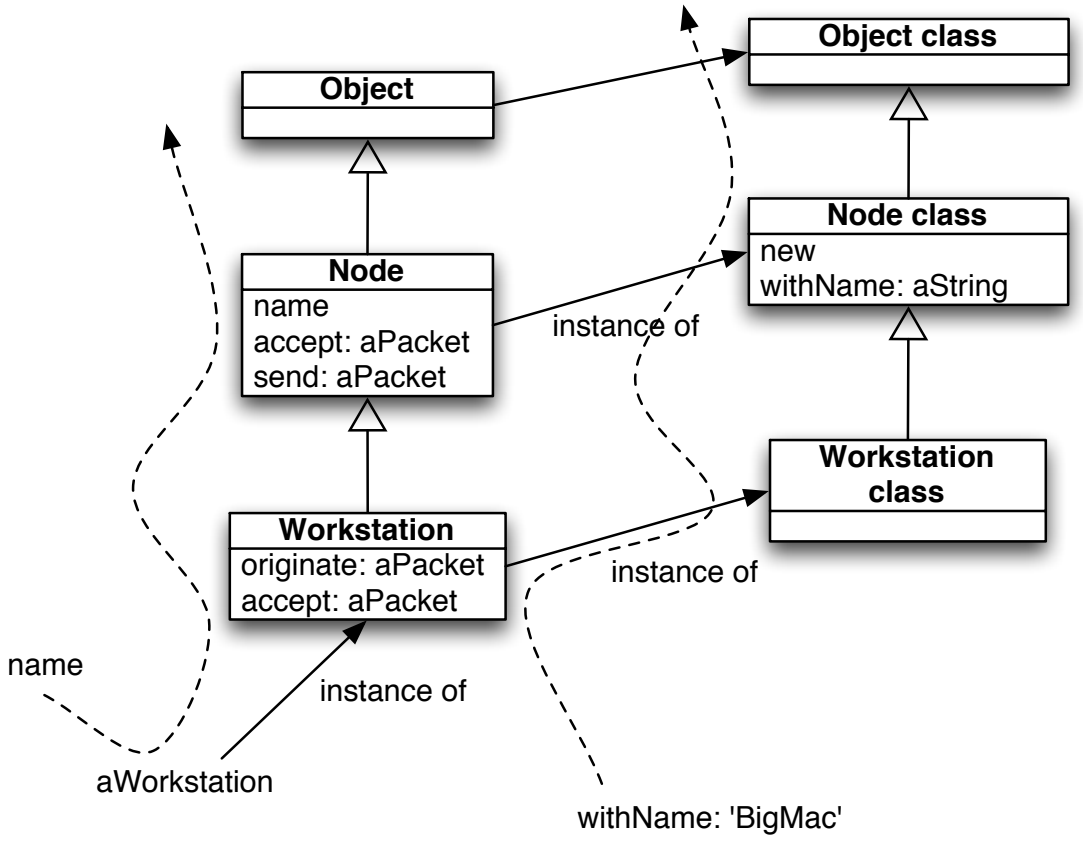
Node withName: #node1
- Node is an instance of
  - "Node class"
- withName: is looked up
  - in the class "Node class"
- withName: defined in
  - "Node class"
- lookup stops +
- method executed



Class Parallel Inheritance



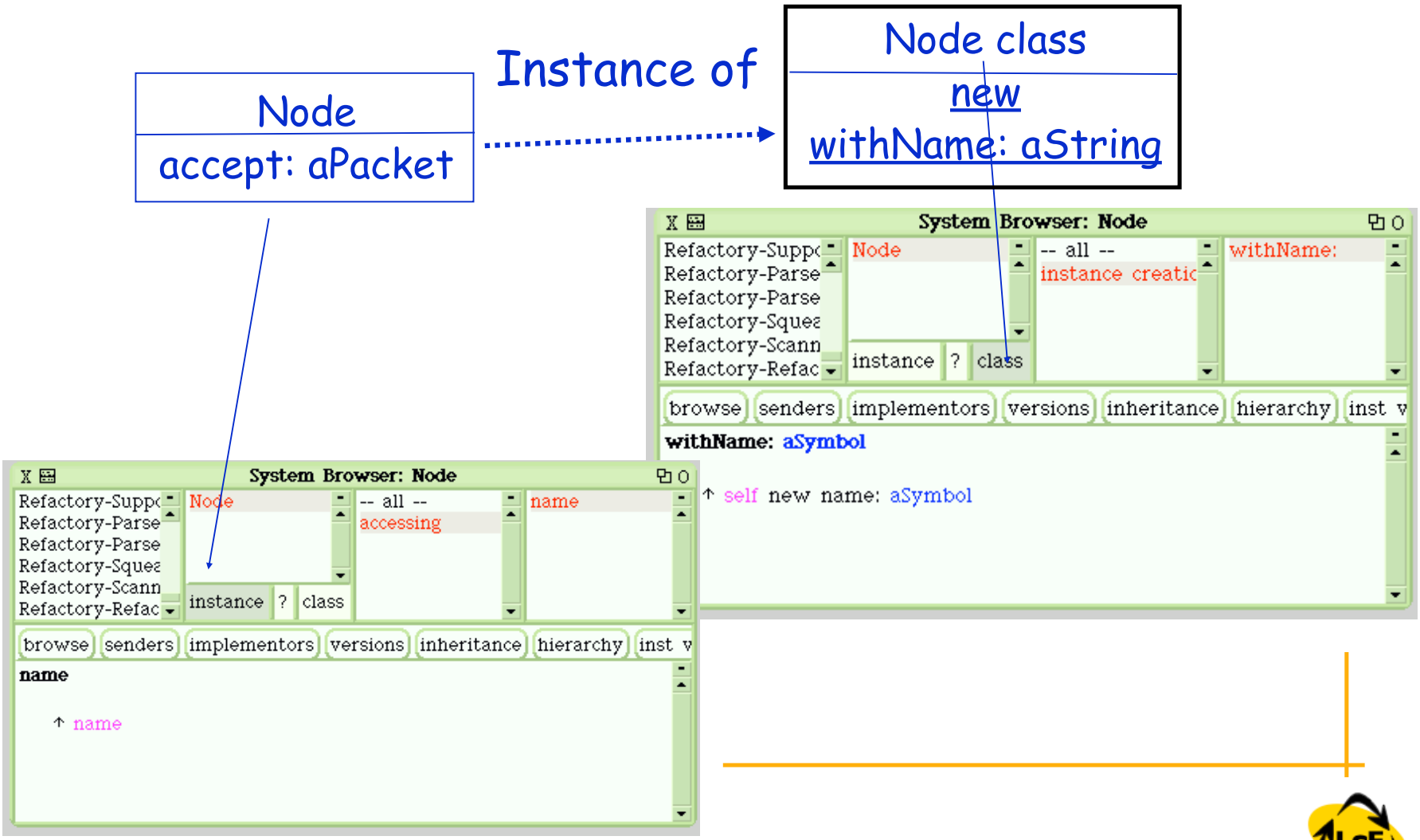
Lookup and Class Methods



Class Parallel inheritance

- Workstation withName: #mac
  - Workstation is an instance of Workstation class
    => withName: is looked up in the class Workstation class
  - withName: is not defined in Workstation class
    => lookup continues in the superclass of Workstation class = Node class
  - withName: is defined in Node class
    => lookup stops + method executed

About the Buttons



Where is new defined?

- Node new: #node1
  - Node is an instance of Node class => new: is looked up in the class Node class
  - new: is not defined in Node class => lookup continues in the superclass of Node class = Object class
  - new: is not defined in Object class => lookup continues in the superclass of Object class ....Class, ClassDescription, Behavior
  - new: is defined in Behavior => lookup stops + method executed.

- This is the same for Array new: 4
  - new: is defined in Behavior (the ancestor of Array class)

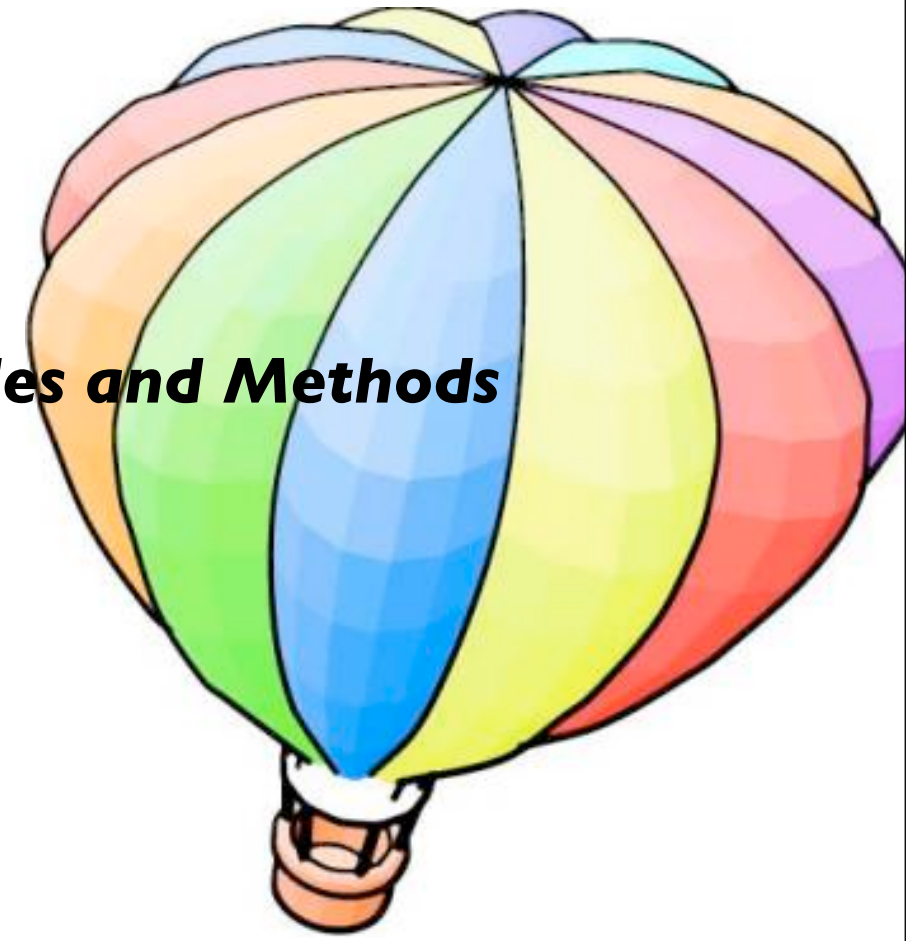
- Hint: Behavior is the essence of a class. ClassDescription represents the extra functionality for browsing the class. Class supports poolVariable and classVariable.

Recap

- Everything is an object
- Each object is instance of one class
- A class (X) is also an object, the sole instance of its associated metaclass named X class
- An object is a class if and only if it can create instances of itself.
- A Metaclass is just a class whose instances are classes
  - Point class is a metaclass as its instance is the class Point

Roadmap

- Indexed Classes
- Classes as Objects
- ***Class Instance Variables and Methods***
- Class Variables



Class Methods

- As any object a (meta)class can have methods that represent the behavior of its instance: a class
- Uniformity => Same rules as for normal classes
- No constraint: just normal methods
- Can only access instance variable of the class:

Class Method Examples

- NetworkManager class>>new can only access uniqueInstance class instance variable and not instance variables (like nodes).

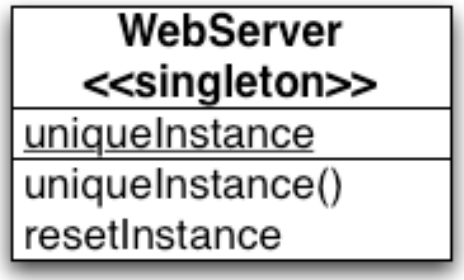
- Default Instance Creation class method:
  - new/new: and basicNew/basicNew: (see Direct Instance Creation)
    - Packet new
  - Specific instance creation method
    - Packet send: 'Smalltalk is fun' to: #lpr

Class Instance Variables

- Like any object, a class is an instance of a class that can have instance variables that represent the state of a class.
- When Point defines the new instance variable z, the instances of Point have 3 value (one for x, one for y, and one for z)
- When a metaclass defines a new instance variable, then its instance (a Class) gets a new value in addition to subclass, superclasses, methodDict…

The Singleton Pattern

- A class having only one instance
- We keep the instance created in an instance variable




```
WebServer class
    instanceVariableNames: 'uniqueInstance'

WebServer class>>new
    self error: 'You should use uniqueInstance to get the unique instance'

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

Singleton

- WebServer being an instance of WebServer class has an instance variable named uniqueInstance.

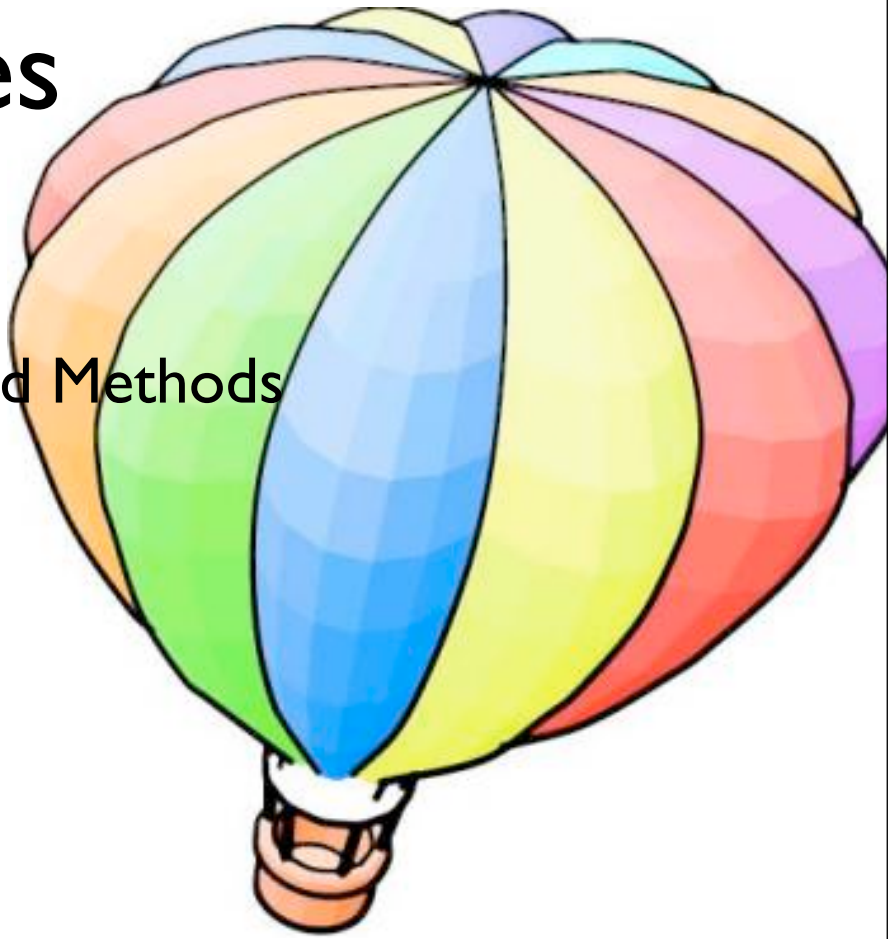
- WebServer has a new value that is associated with uniqueInstance

Design Implications

- An instance variable of a class can be used to represent information shared by all the instances of the class. However, you should use class instance variables to represent the state of the class (like the number of instances, ...) and not information of its instance.
- Should use shared Variable instead (next Section).

Advanced Classes

- Indexed Classes
- Classes as Objects
- Class Instance Variables and Methods
- ***Class Variables***



classVariable = Shared Variables

- How to share state between all the instances of a class:
  Use a classVariable

- a classVariable is shared and directly accessible by all the instances of the class and subclasses
- A pretty bad name: should have been called Shared Variables (now fixed in VW)
- Shared Variable => begins with an uppercase letter
- a classVariable can be directly accessed in instance methods and class methods

classVariable = shared Variab. (Sq)

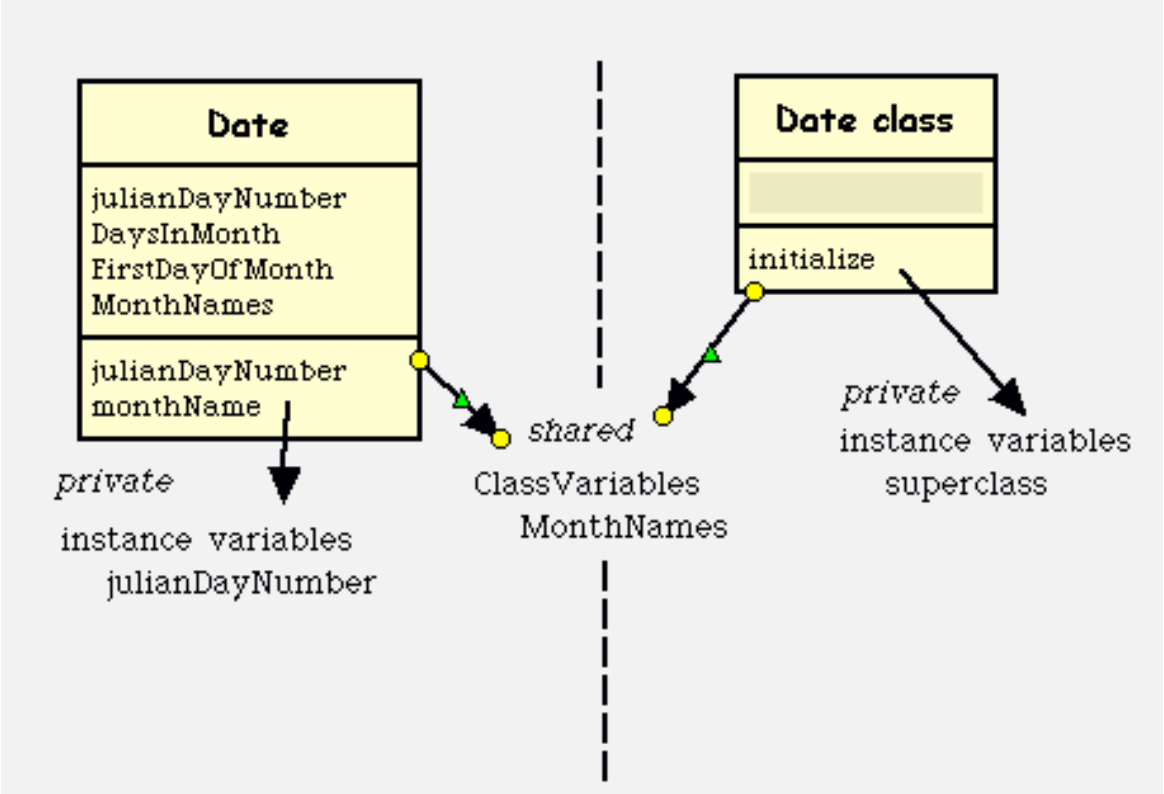
```
Magnitude subclass: #Date
   instanceVariableNames: 'julianDayNumber '
   classVariableNames: 'DaysInMonth FirstDayOfMonth MonthNames
SecondsInDay WeekDayNames '
   poolDictionaries: ''
   category: 'Kernel-Magnitudes'
```

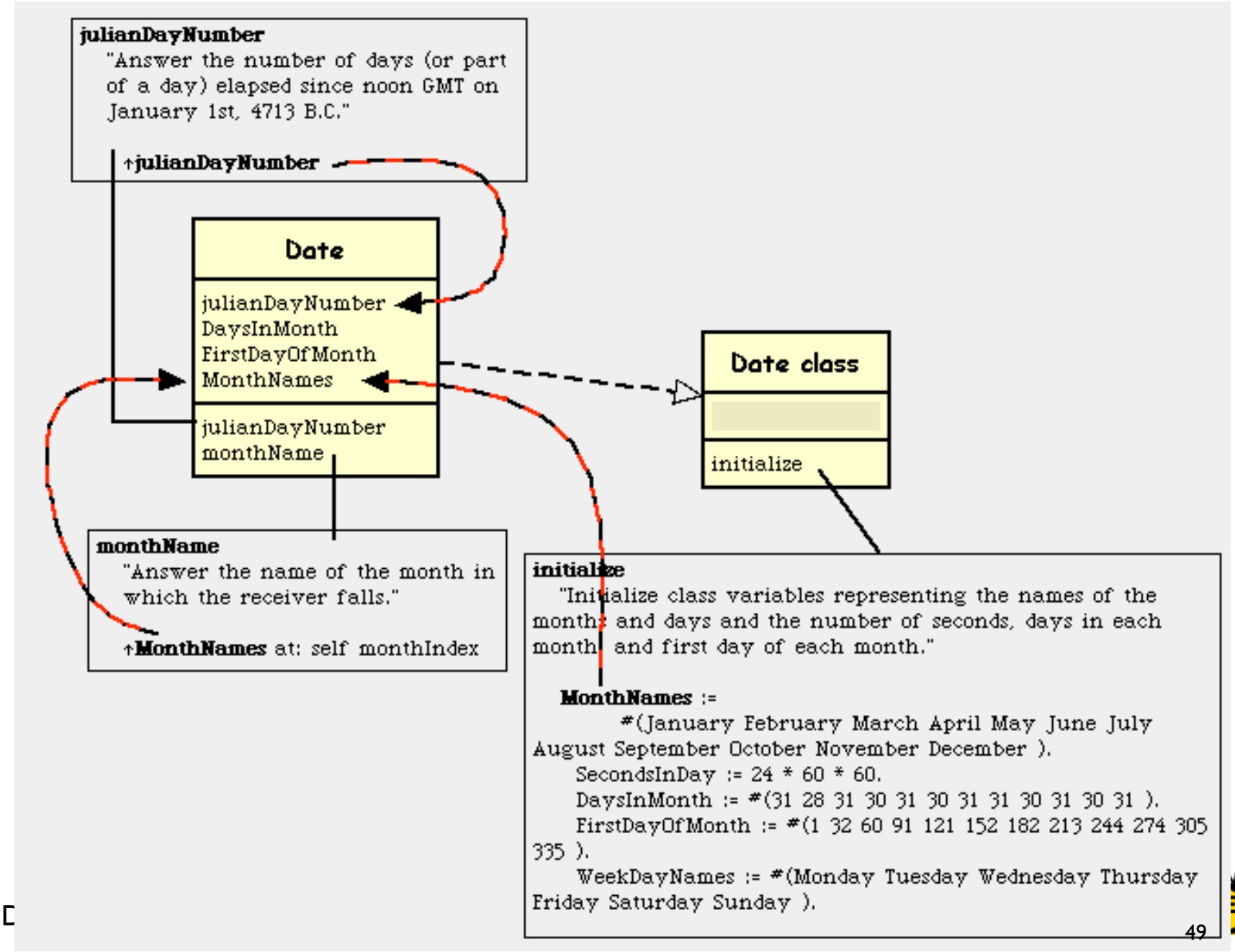
Date class>>initialize

"Initialize class variables representing the names of the months and days and the number of seconds, days in each month, and first day of each month."

```
MonthNames  := #(January February March April May June July August
September October November December ).
SecondsInDay := 24 * 60 * 60.
DaysInMonth := #(31 28 31 30 31 30 31 31 30 31 30 31 ).
FirstDayOfMonth := #(1 32 60 91 121 152 182 213 244 274 305 335 ).
WeekDayNames := #(Monday Tuesday Wednesday Thursday Friday Saturday
Sunday ).
```

ClassVariable vs. Instance Variables

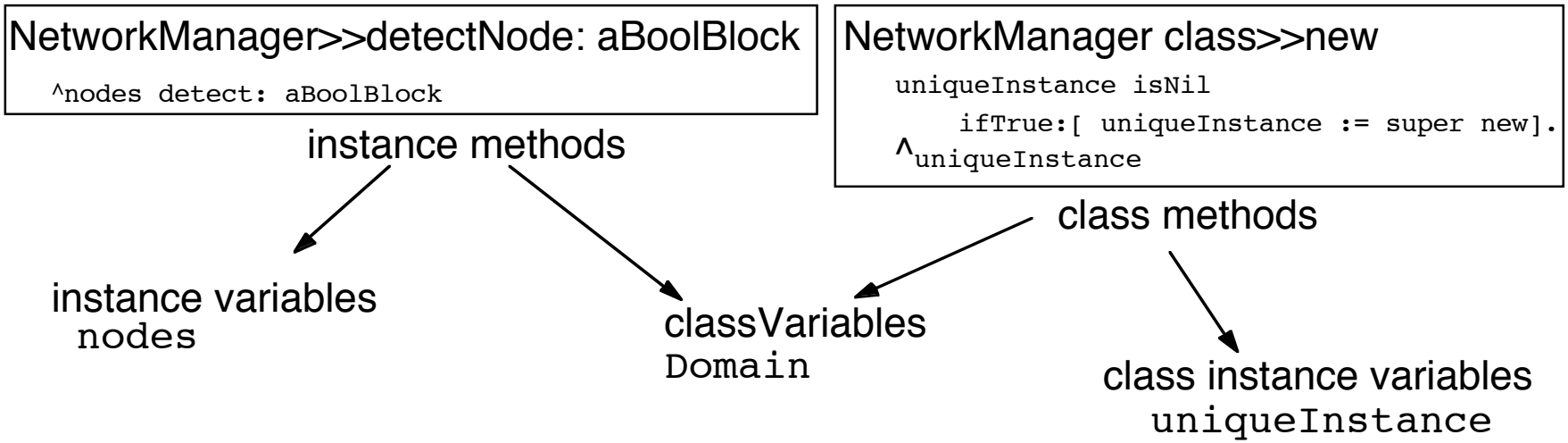



julianDayNumber
"Answer the number of days (or part of a day) elapsed since noon GMT on January 1st, 4713 B.C."
↑julianDayNumber
Date
julianDayNumber
DaysInMonth
FirstDayOfMonth
MonthNames
julianDayNumber
monthName
Date class
initialize
monthName
"Answer the name of the month in which the receiver falls."
↑MonthNames at: self monthIndex
initialize
"Initialize class variables representing the names of the months and days and the number of seconds, days in each month, and first day of each month."
MonthNames :=
#(January February March April May June July August September October November December).
SecondsInDay := 24 * 60 * 60.
DaysInMonth := #(31 28 31 30 31 30 31 31 30 31 30 31).
FirstDayOfMonth := #(1 32 60 91 121 152 182 213 244 274 305 335).
WeekDayNames := #(Monday Tuesday Wednesday Thursday Friday Saturday Sunday).

Class Instance Variables vs classVariables

- a classVariable is shared and directly accessible by all the instances and subclasses
- Class instance variables, just like normal instance variables, can be accessed only via class message and accessors:
  - an instance variable of a class is private to this class.
- Take care: when you change the value of a classVariable the whole inheritance tree is impacted!

Summary of Variable Visibility



ClassVariables...

- ClassVariables can be used in conjunction with instance variables to cache some common values that can be changed locally in the classes.

Example

- in the Scanner class a table describes the types of the characters (strings, comments, binary....). The original table is stored into a classVariable, its value is loaded into the instance variable. It is then possible to change the value of the instance variable to have a different scanner.

```
Object subclass: #Scanner
    instanceVariableNames: 'source mark prevEnd hereChar
    token tokenType buffer typeTable '
    classVariableNames: 'TypeTable '
    category: 'System-Compiler-Public Access'
```

What you should know

- Classes are objects too
- Class methods are just methods on objects that are classes
- Classes are also represented by instance variables (class instance variables)
- (Shared Variables) ClassVariables are shared among subclasses and classes (metaclass)