What if classes are objects too

Stéphane Ducasse
Stephane.Ducasse@univ-savoie.fr
http://www.iam.unibe.ch/~ducasse/

Outline

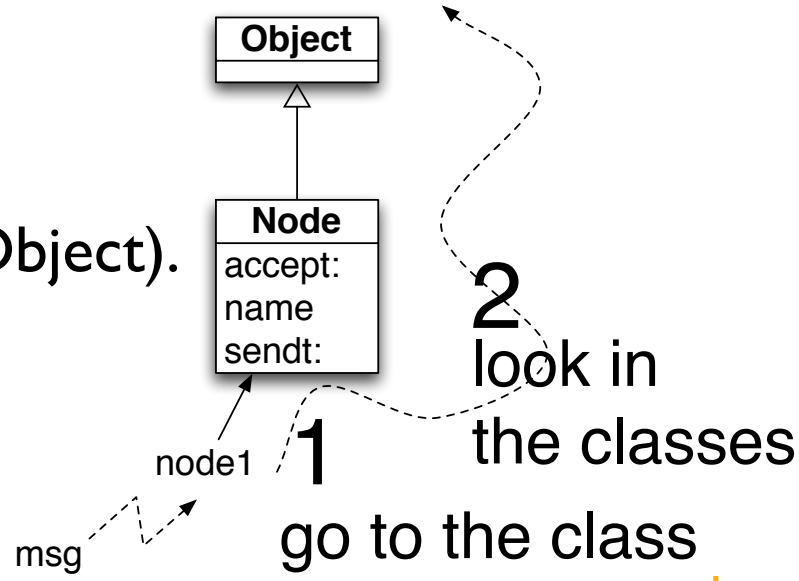
- What is the meaning of
  - instance
  - inheritance?
- Classes as Objects
- Class Instance Variables and Methods
- Class Variables

The Meaning of Is-a

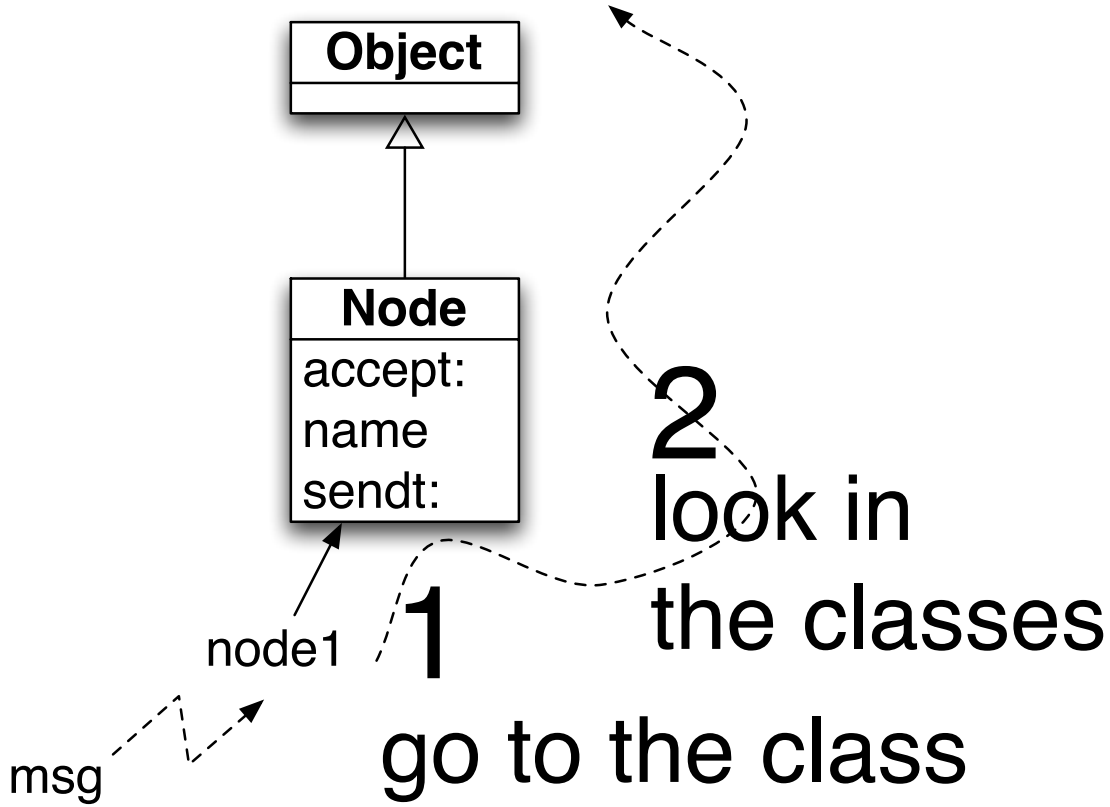
- A class defines the structure and the behavior of all its instances.
- Each instance possesses its own set of values.
- Instances share the behavior defined in their class with other instances via the instance of link.

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
2
look in
the classes
1
go to the class

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

Outline

- What is the meaning of
  - instance
  - inheritance?
- 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

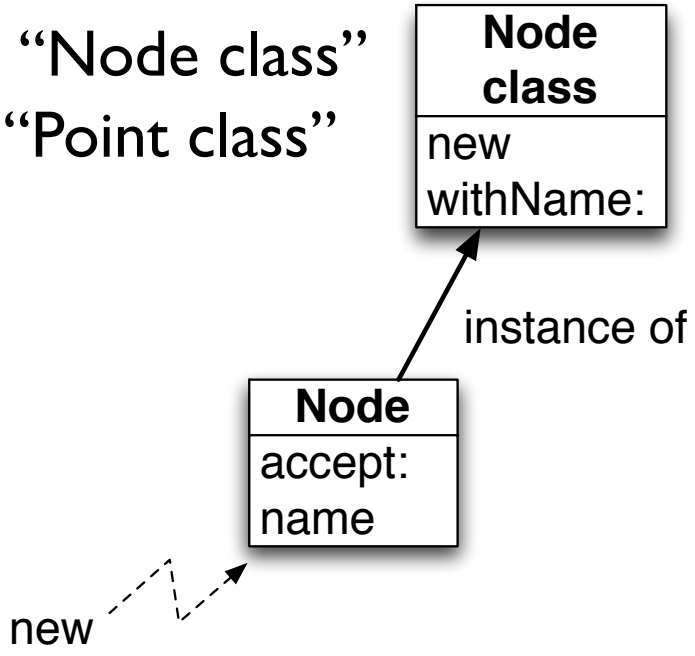
- In some language classes are not objects, still understanding it in Smalltalk will force you to really understand what instance/inheritance means

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…

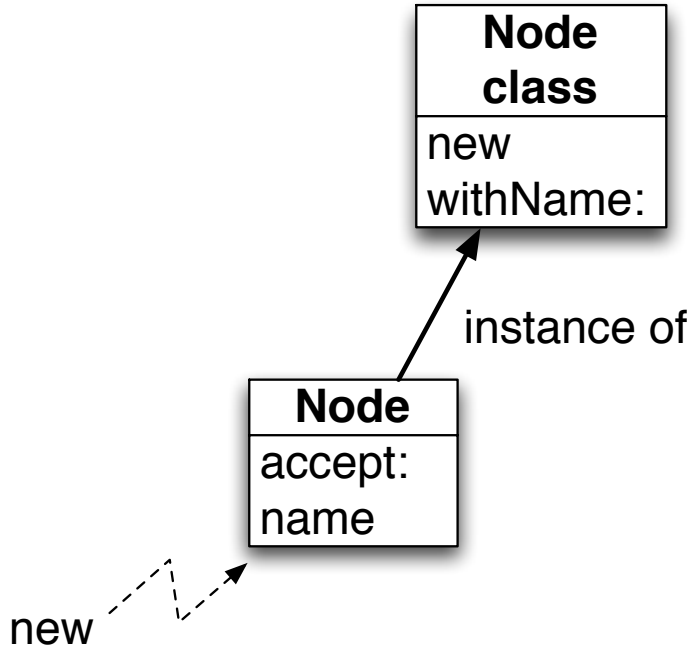
- Every class (X) is the unique instance of its associated metaclass named X class
- Example:
- Node is the unique instance of "Node class"
- Point is the unique instance of "Point class"



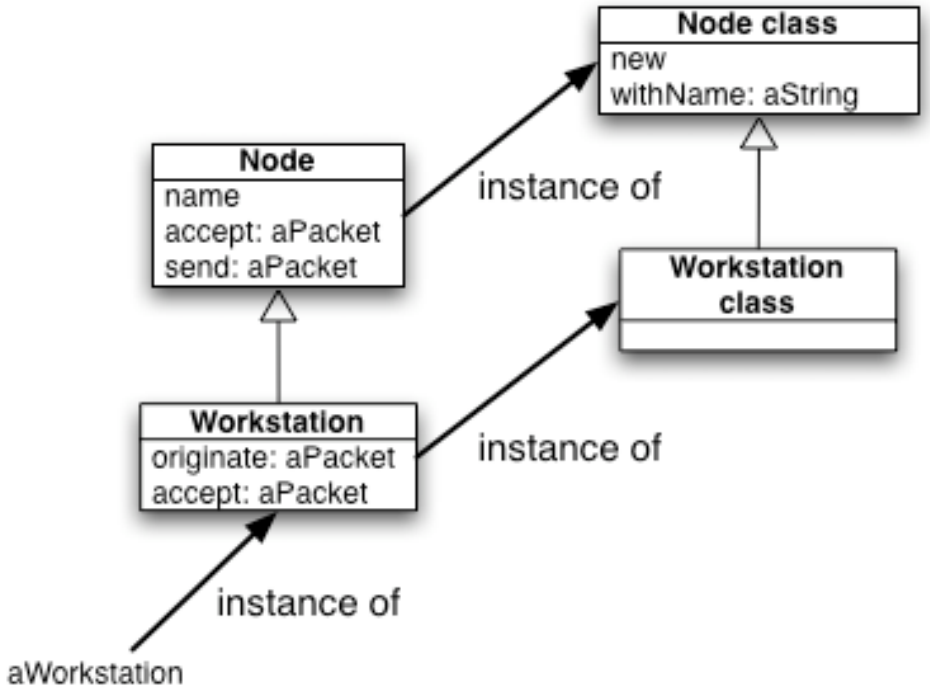
A Class is an Object too…

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

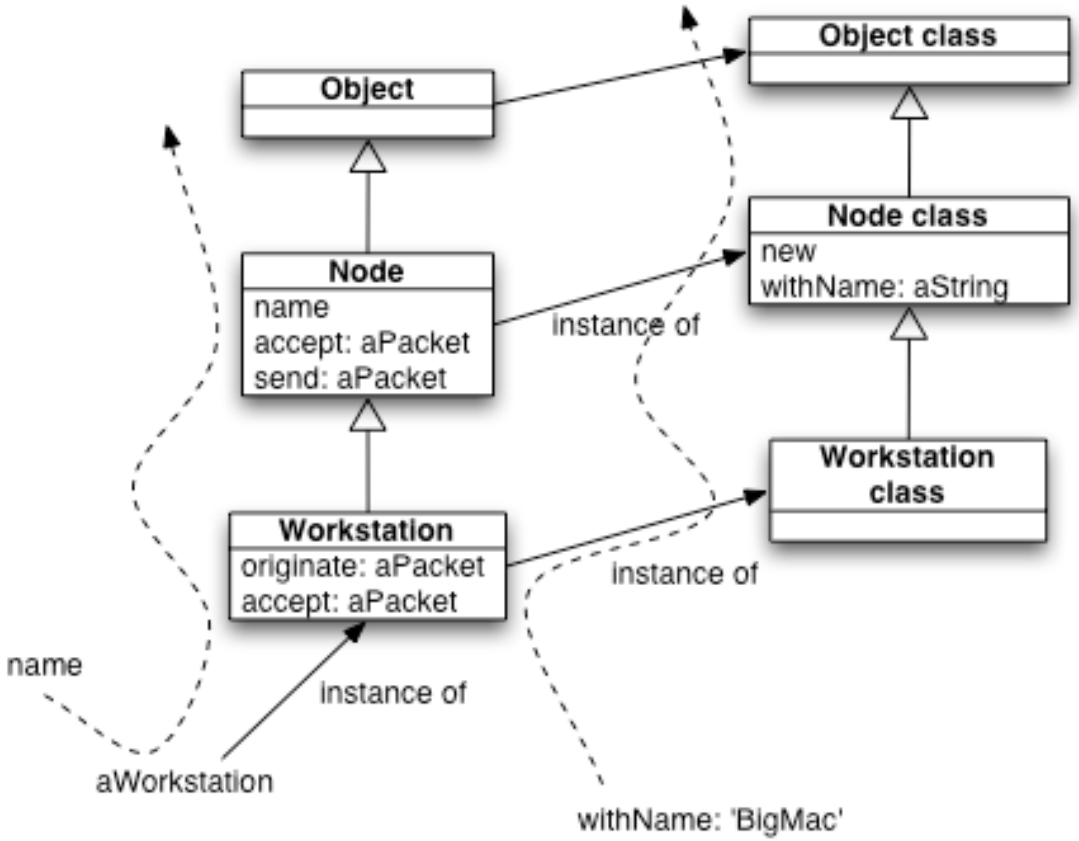
Node withName: #node1
- Node is an instance of
  - "Node class"
- withName: is looked up
  - in the class "Node class"
- withName: defined in
  - "Node class"
- lookup stops +
- method executed



Class Parallel Inheritance



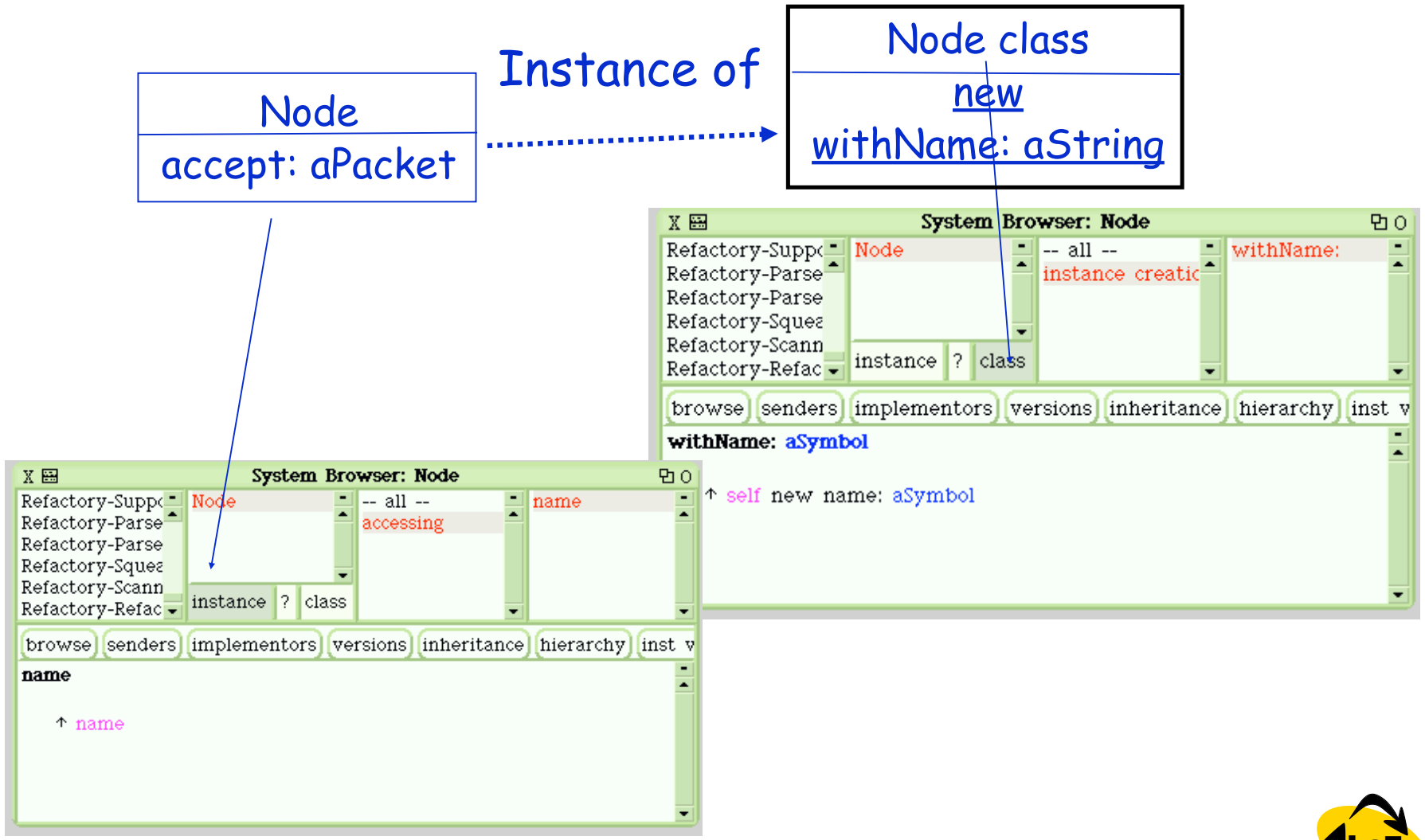
Lookup and Class Methods



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

Outline

- What is the meaning of
  - instance
  - inheritance?
- 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

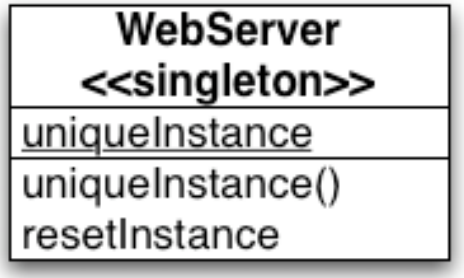
Class Method Examples

- 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

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

Summary

A class is also an object

new is simply a class method

Class methods are looked up the same way instance method:

(1) follow the ***instance-of*** link and go in the class of the receiver

(2) look in method dictionary of class and its superclasses