Inheritance In Pharo in A Nushell

Stéphane Ducasse and Damien Cassou

http://stephane.ducasse.free.fr/ stephane.ducasse@inria.fr damien.cassou@inria.fr

In A Nutshell

- Single inheritance
  - Inheritance of instance variables -> class definition time
  - Inheritance of behavior -> runtime
- Only virtual calls
- Method lookup starts in the class of the receiver
- `self` represents the receiver, `super` too
- Class methods are virtual too
- Messages not understood is a message and a hook of the metaobject protocol

Root of Inheritance

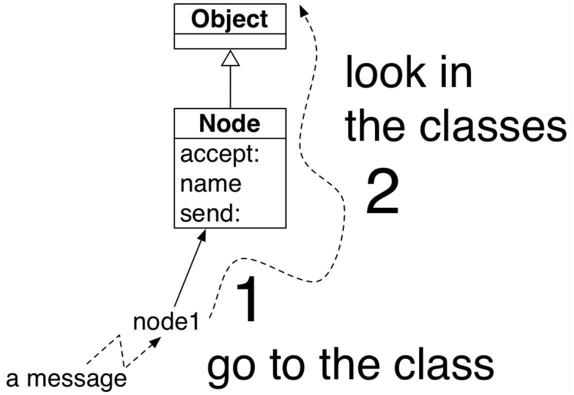
- `Object` is the root of most classes
- `ProtoObject` (the superclass of `Object`) is for special purposes
  - `ProtoObject`'s goal in life is to raise errors to most of the messages
  - This is important to build proxies

Inheritance of Instance Variables

- Inheritance of instance variables is made at class definition time
    - The instance variables of a new class are computed based on its own instance variables and the ones of its superclass
    - This happens at class definition time

Inheritance of Behavior and the Lookup

- Inheritance of behavior is dynamic and done at runtime
- The *method* corresponding to the *message* is *looked up*
  - starting from the class of the receiver
  - if not found there, the **lookup** follows the inheritance chain

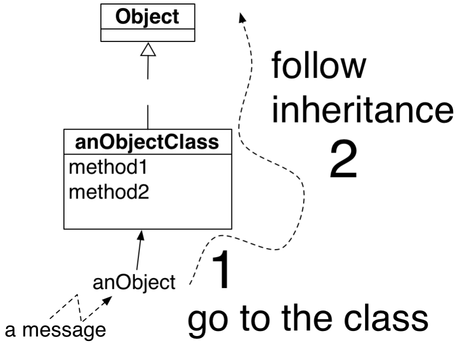


self and super

- `self` and `super` represents the receiver of the message (as in Java, C#...)
- `super` is used to access overriden methods

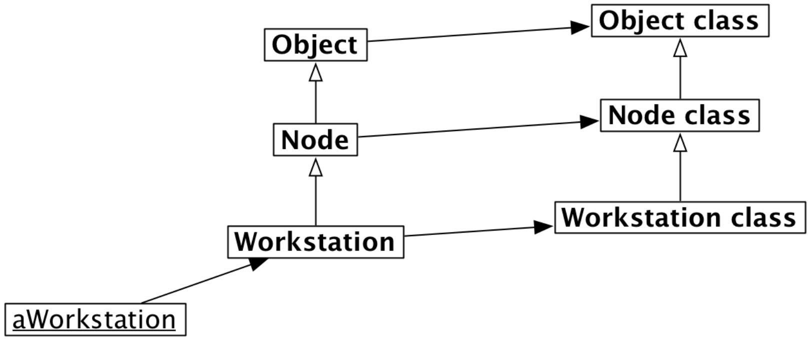
Lookup of Class Methods is No Different

- Sending a message to a class is also late-bound (dynamically resolved)
  - contrary to Java/C#
  - no `static`, only methods of another object (a class)
- Only one rule:
  - when a message is sent to an object, a method is searched starting from the class of the object and following the inheritance chain

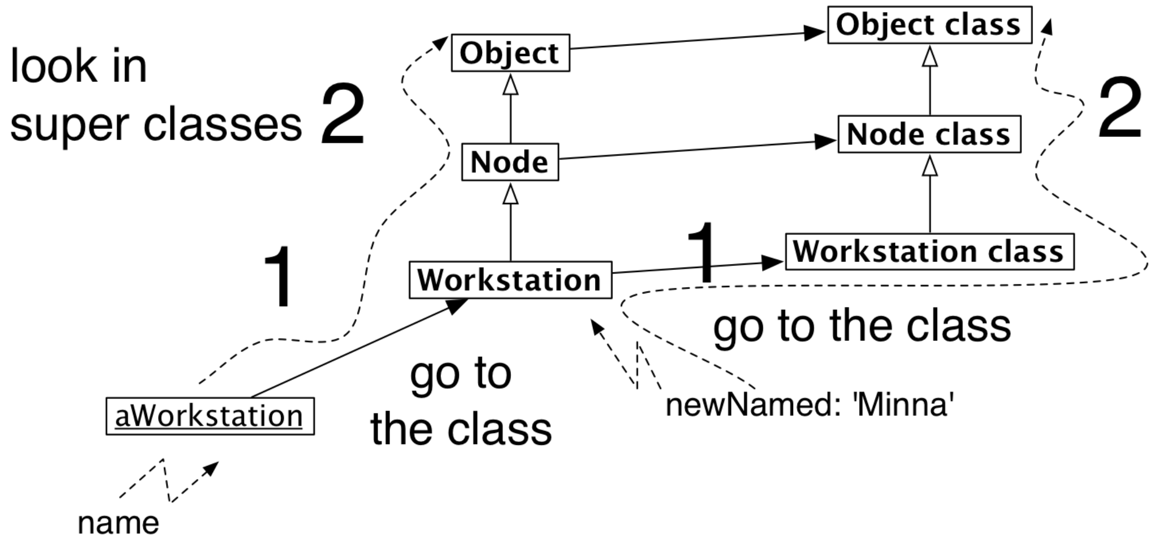


A Class is an Instance of Another Class

- A class `X` is always the unique instance of another class `X class`
  - The class of `Node` is `Node class`

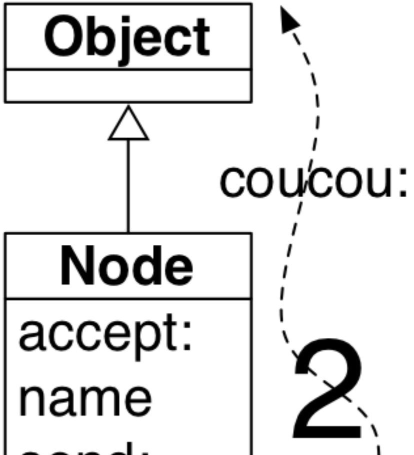


Lookup of Class Methods is No Different



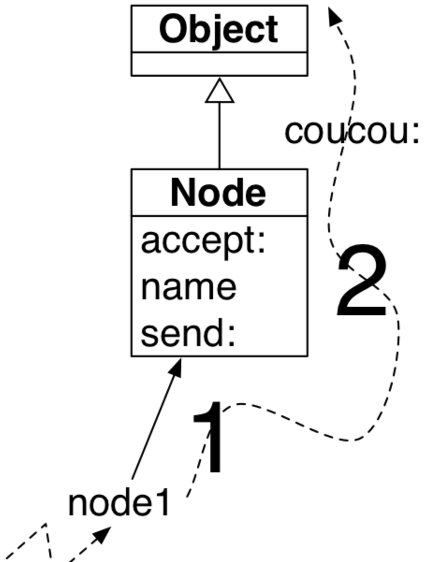
When Message is Not Found

- If no method is found in the topmost superclass (Object class):
  - a message #doesNotUnderstand: is sent to the original receiver
  - this message includes the original one



Step by Step...

- node1 coucou: #stef
  1. coucou: is looked up in Node
  2. not defined in Node -> lookup continues in Object
  3. not defined in Object => system sends doesNotUnderstand: to node1
  4. doesNotUnderstand: is looked up in Node
  5. not defined in Node -> lookup continues in Object
  6. Object>>doesNotUnderstand: is found and executed



doesNotUnderstand: is a Message

- `doesNotUnderstand:` is a message
- Every class can customize error handling
- Important hook for automatic delegation
  - when an object behaves the same way as its target

```
Proxy>>doesNotUnderstand: aMessage
  "Delegate aMessage to my target"
  ^ aMessage sendTo: target
```

doesNotUnderstand: and the Debugger

What happens when a message is not understood can be customized:

- the message `doesNotUnderstand:` is looked up
- when no class redefines the message `doesNotUnderstand:`
  - a `MessageNotUnderstood` exception is raised
  - when there is no handler of that exception, the default is to open a debugger
- this behavior can be customized to hide/control/log errors

```
SomeClass>>doSomething
  [ ... ]
  on: MessageNotUnderstood
  do: [ Transcript show: 'Something went wrong with a message' ]
```

What you should Know

- Inheritance of instance variables -> class definition time
- Inheritance of behavior -> at runtime.
- `self` *always* represents the receiver, the method lookup starts in the class of the receiver
- `super` *always* represents the receiver but method lookup starts in the superclass of the class using it
- `doesNotUnderstand:` is a message and a hook of the metaobject protocol.