Inheritance and Lookup

3: Super

Stéphane Ducasse and Damien Cassou

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

Goal

- Sending a message
- Dynamic binding/method lookup
- `super` semantics and the differences with `self`

What is super?

Take 5 min and write the definition of super?

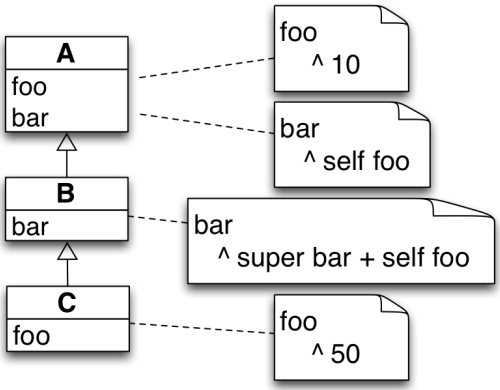
- Your definition should have two points:
  - What does it represent?
  - How the method are looked up when a message is sent to `super`?

How do we access to an overriden method?

- You define a method with the same name that one in an upper class.
- You want to execute it in your subclass?
- Use `super` instead of `self`

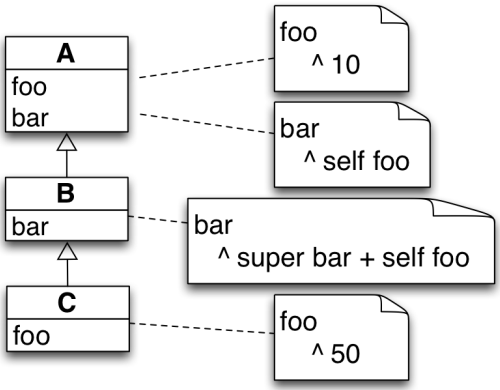
```
Workstation>>accept: aPacket
    (aPacket isAddressedTo: self)
        ifTrue: [ Transcript show: 'Accepted by the Workstation ', self name asString ]
        ifFalse: [ super accept: aPacket ]
```

Challenge yourself with super!



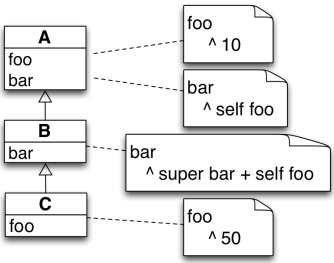

```
A new bar
> ...
B new bar
> ...
C new bar
> ...
```

Challenge yourself with super!




```
A new bar
> 10
B new bar
> 20
C new bar
> 100
```

Super changes the class where the lookup starts



C new bar
> 100

- bar is sent to aC (an instance of C): bar is looked up in C, not found, look in B
- bar is found in class B, and applied to the receiver aC .
- bar is sent to super ,
- super is the receiver (aC), but lookup starts above class B
- bar is found in class A and it is applied to the receiver aC .
- foo is sent to self : self represents the receiver: aC
- foo is found in class C and applied to aC , it returns 50.

Super?

- `super` refers to the receiver of the message (like `self`. Yes!)
- The method lookup starts in the superclass of ...?

Super starts lookup in superclass of the class using it

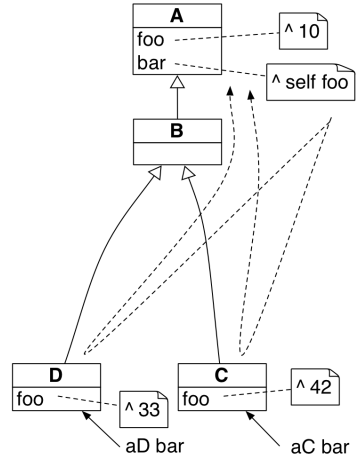
- `super` refers to the receiver of the message (like `self`. Yes!)
- The method lookup starts in the superclass of the class containing the super expression.

Super is static / self is dynamic

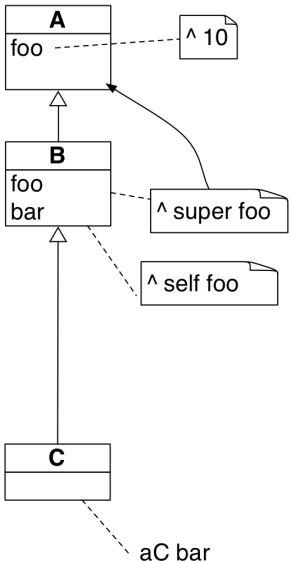
- There is no reference to the receiver in the method lookup of a `super` invocation!

self is dynamic

- When we read the body of method `bar`, there is no way that we know which method `foo` will be executed.
- New instances of different classes can be created and the message `bar` sent to them.
- `self` acts as a hook. Code of subclasses can be injected into `self` sends.



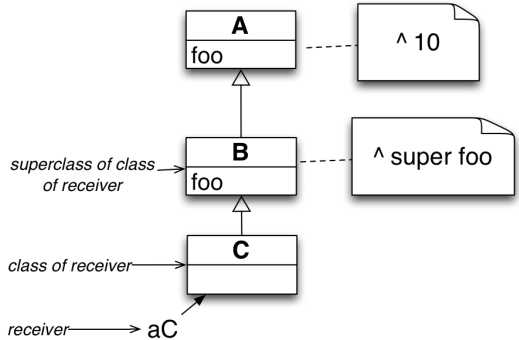
super is static



- At compilation-time, just reading the code we know that we should look above the class containing the **method** using `super`.

Yes even some books got it wrong

- Wrong definition: super looks for the method in the superclass of the class of the **receiver**.
- Wrong!
- It would loop forever!
- `aC foo` loops, because `super` points to `aC` and the superclass of the class of the receiver is B.



What you should know

- `self` *always* represents the receiver
- the method lookup maps a message to a method
- the method lookup starts in the class of the receiver...
- ...and goes up in the hierarchy
- `super` is the receiver, lookup starts is superclass of the method using the expression.
- self sends act as a hook. Code of subclasses may be invoked.