

Inheritance Semantics and Lookup

Inheritance

- Do not want to rewrite everything!
- Often times want small changes
- Class hierarchies for sharing of definitions
- Each class defines or refines the definition of its ancestors
- => inheritance

Inheritance

New classes

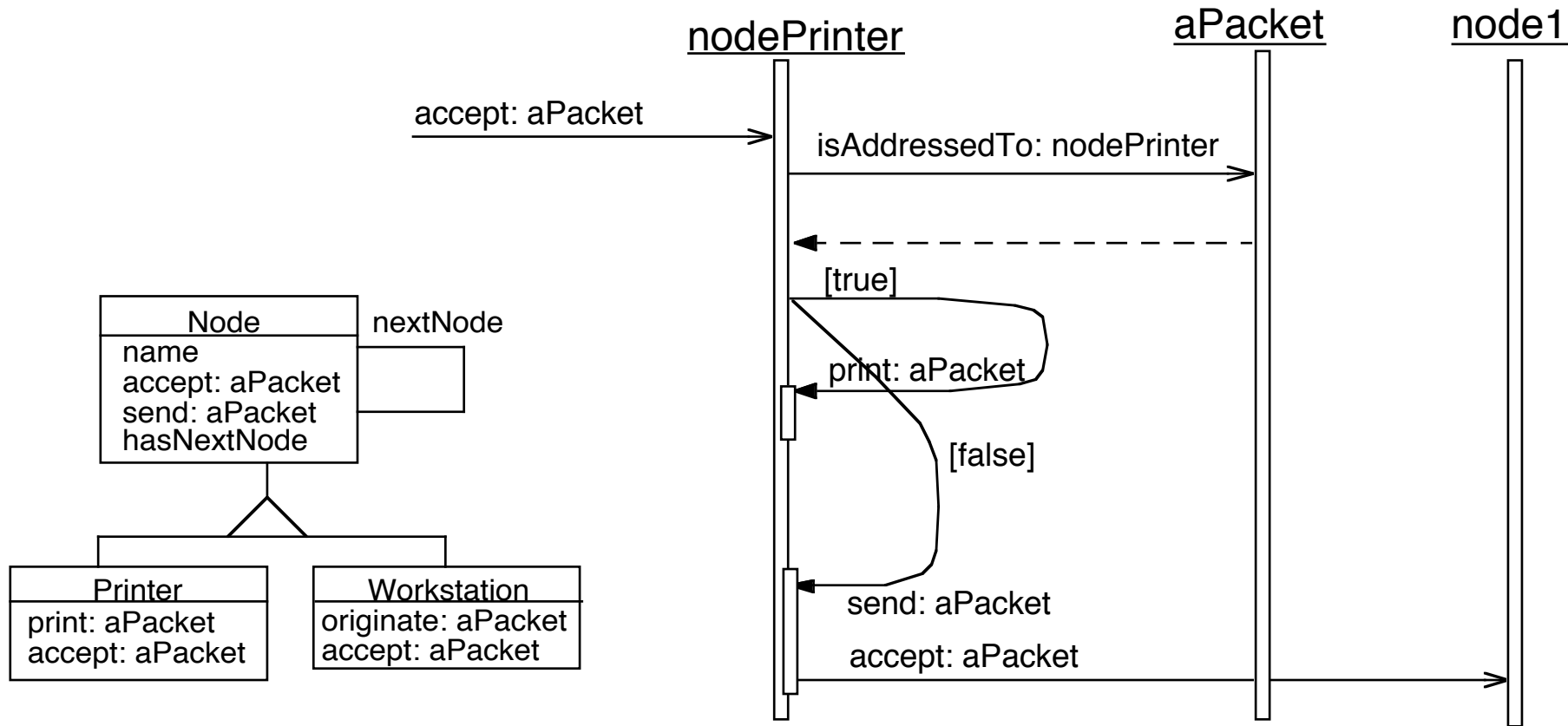
- Can add state and behavior
- Can specialize ancestor behavior
- Can use ancestor's behavior and state
- Can redefine ancestor's behavior

to existing ones

Inheritance in Smalltalk

- Single inheritance
- Static for the instance variables.
- At class creation time the instance variables are collected from the superclasses and the class. No repetition of instance variables.
- Dynamic for the methods.
- Late binding (all virtual) methods are looked up at run-time depending on the dynamic type of the receiver.

Remember...



Node

```
Object subclass: #Node
  instanceVariableNames: 'name nextNode '
  ...
```

```
Node>>accept: aPacket
```

```
"Having received the packet, send it on. This is the default behavior
subclasses will probably override me to do something special."
```

```
self hasNextNode if True: [self send: aPacket]
```

```
Node>>send: aPacket
```

```
"Precondition: there is a next node. Send a packet to the next node."
```

```
self nextNode accept: aPacket
```

Workstation

```
Node subclass: #Workstation
    instanceVariableNames: ''
    ...
```

```
Workstation>>accept: aPacket
```

“when a workstation accepts a packet addressed to it, it prints some trace on the transcript”

```
    (aPacket isAddressedTo: self)
        ifTrue:[ Transcript show: 'A packet is accepted
by the Workstation ', self name asString]
        ifFalse: [super accept: aPacket]
```

```
Workstation>>originate: aPacket
```

```
    aPacket originator: self.
    self send: aPacket
```

Message Sending & Method Lookup

receiver selector args

- Looking up a method: When a message (receiver selector args) is sent, the method corresponding to the message selector is looked up through the inheritance chain.
- The lookup starts in the *CLASS* of the *RECEIVER*.
- If the method is defined in the class dictionary, it is returned.
- Otherwise the search continues in the superclasses of the receiver's class. If no method is found and there is no superclass to explore (class *Object*), a new method called *#doesNotUnderstand:* is sent to the receiver, with a representation of the initial message.

In Smalltalk

- If no method is found and there is no superclass to explore (class Object), a new method called `#doesNotUnderstand:` is sent to the receiver, with a representation of the initial message.

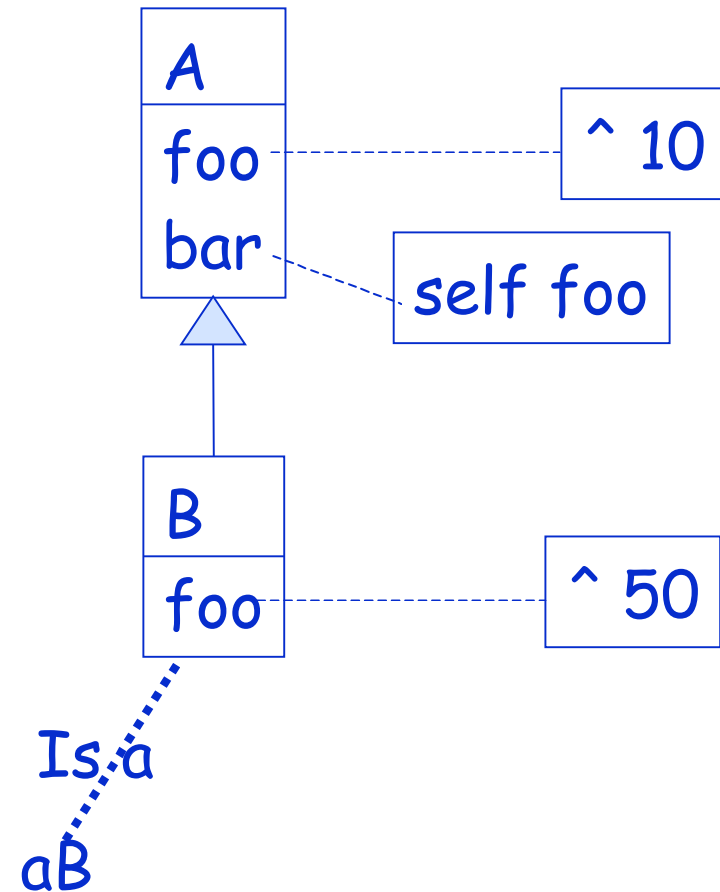
Method Lookup starts in Receiver Class

aB foo

- (1) aB class => B
- (2) Is foo defined in B?
- (3) Foo is executed -> 50

aB bar

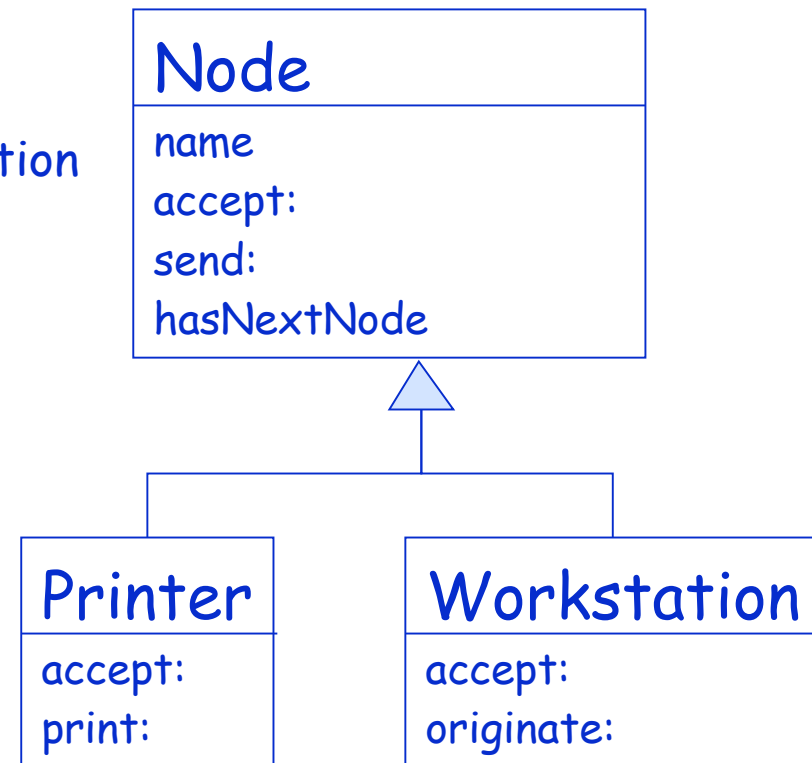
- (1) aB class => B
- (2) Is bar defined in B?
- (3) Is bar defined in A?
- (4) bar executed
- (5) Self class => B
- (6) Is foo defined in B?
- (7) Foo is executed -> 50



Method Lookup Examples

- node1 accept: aPacket
 - node1 is an instance of Node
 - accept: is looked up in the class Node
 - accept: is defined in Node => lookup stops + method executed

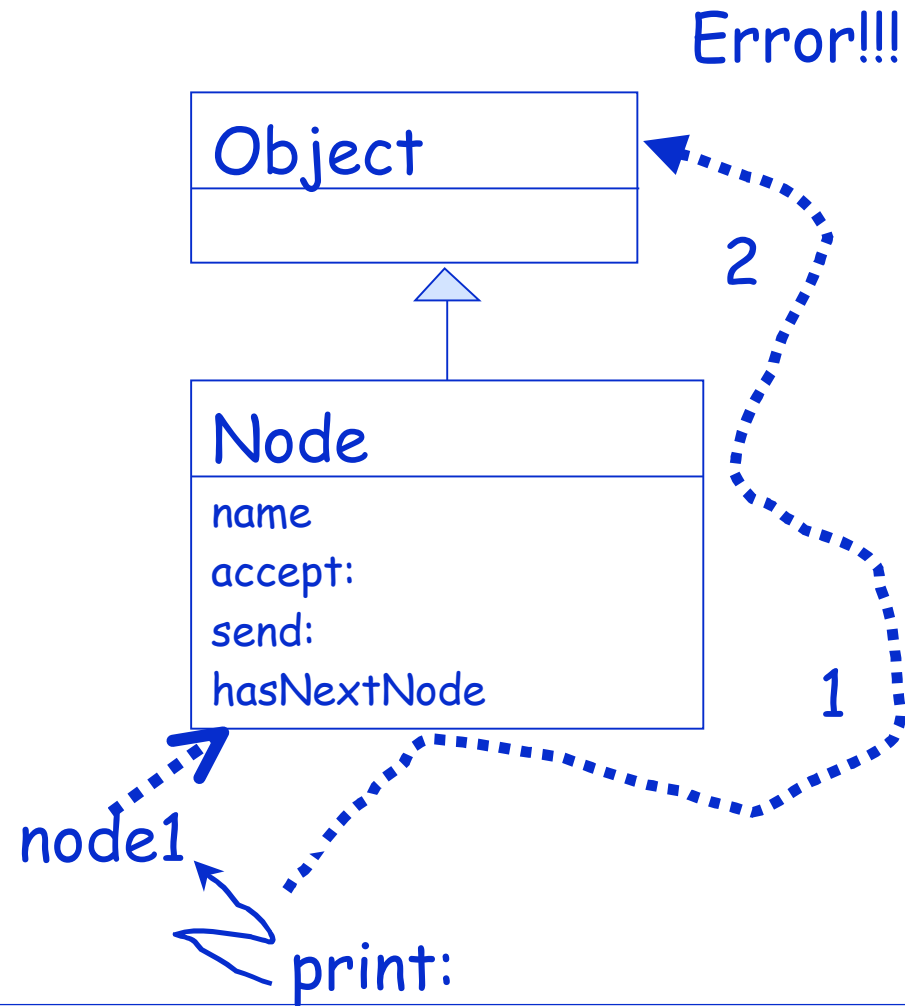
- macNode accept: aPacket
 - macNode is an instance of Workstation
 - accept:
is looked up in the class Workstation
 - accept: is defined in Node
=> lookup stops + method executed



Method Lookup Examples (II)

- `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
- `node1 print: aPacket`
 - `node` is an instance of `Node`
 - `print:` is looked up in the class `Node`
 - `print:` is not defined in `Node` => lookup continues in `Object`
 - `print:` is not defined in `Object` => lookup stops + exception

Graphically...

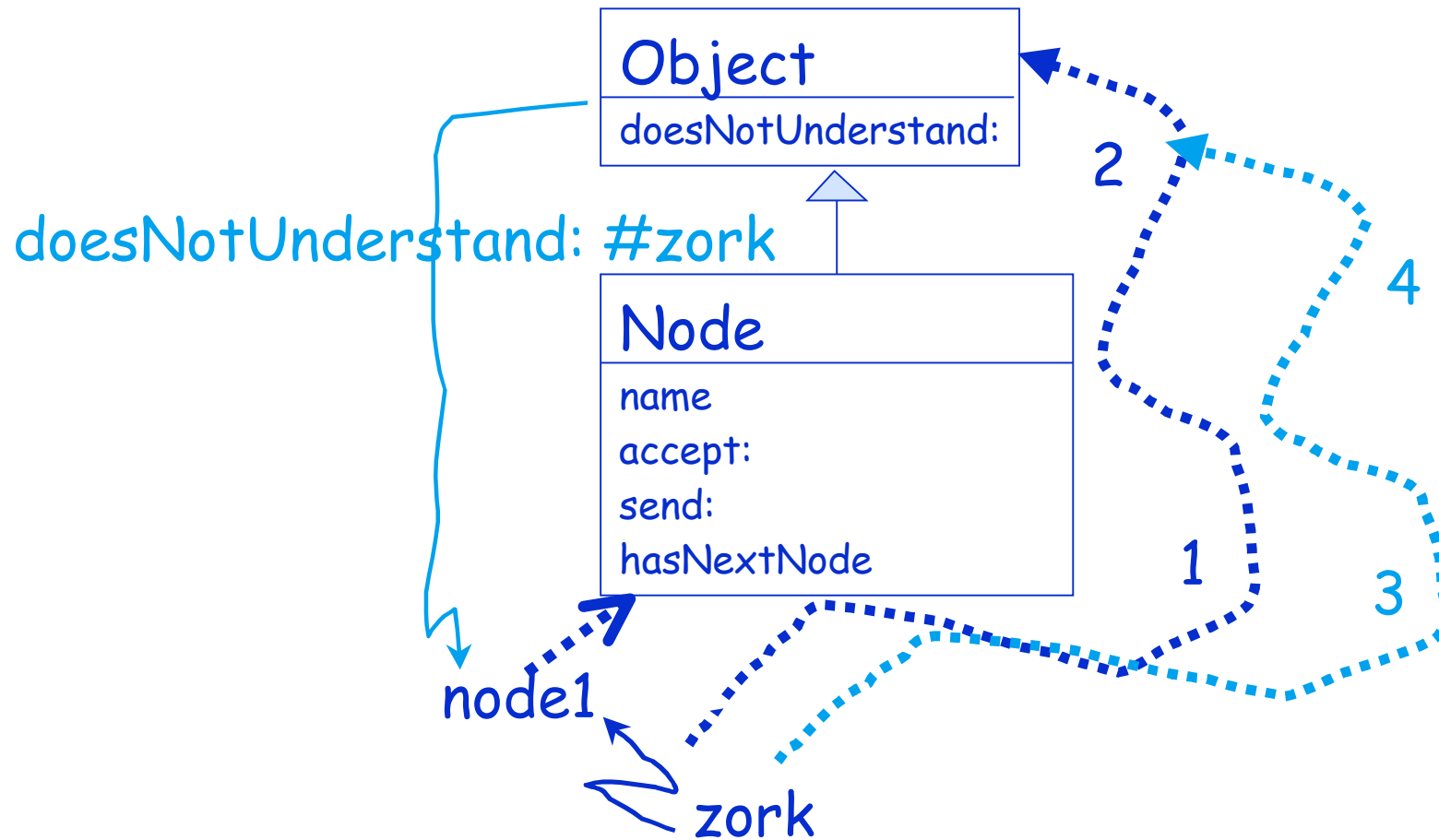


...in Smalltalk

- node1 print: aPacket
 - node is an instance of Node
 - print: is looked up in the class Node
 - print: is not defined in Node > lookup continues in Object
 - print: is not defined in Object => lookup stops + exception

- message: node1 doesNotUnderstand: #(#print aPacket) is executed
- node1 is an instance of Node so doesNotUnderstand: is looked up in the class Node
- doesNotUnderstand: is not defined in Node => lookup continues in Object
- doesNotUnderstand: is defined in Object => lookup stops + method executed (open a dialog box)

Graphically...



self **always** represents the receiver

A new foo

-> 10

B new foo

-> 10

C new foo

-> 50

A new bar

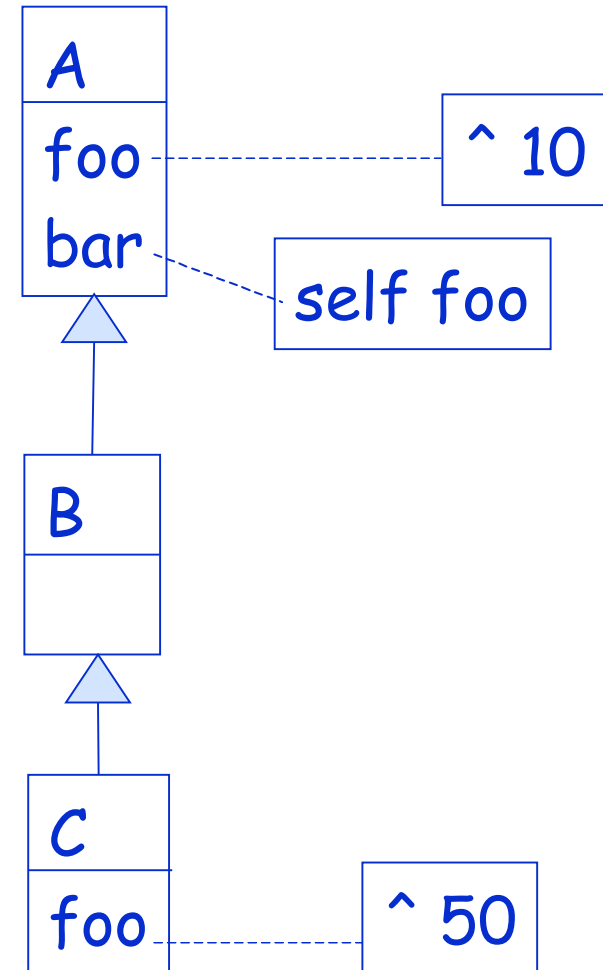
-> 10

B new bar

-> 10

C new bar

-> 50



How to Invoke Overridden Methods?

Solution: Send messages to super

When a packet is not addressed to a workstation, we just want to pass the packet to the next node, i.e., we want to perform the default behavior defined by Node.

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

Design Hint: Do not send messages to super with different selectors than the original one. It introduces implicit dependency between methods with different names.

The semantics of super

- Like `self`, `super` is a pseudo-variable that refers to the receiver of the message.
- It is used to invoke overridden methods.
- When using `self`, the lookup of the method begins in the class of the receiver.
- When using `super`, the lookup of the method begins in the superclass of the class of the method containing the `super` expression and NOT in the superclass of the receiver class.
- This means, `super` causes the method lookup to begin searching in the superclass of the class of the method containing `super`

super changes lookup starting point

A new bar

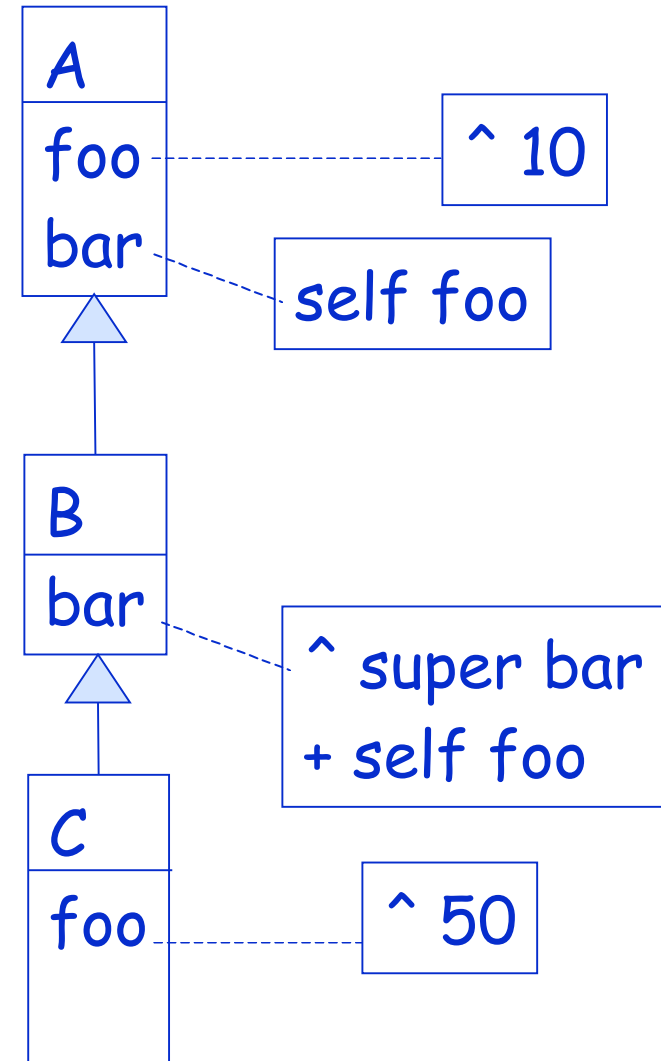
-> 10

B new bar

-> 10 + 10

C new bar

-> 50 + 50

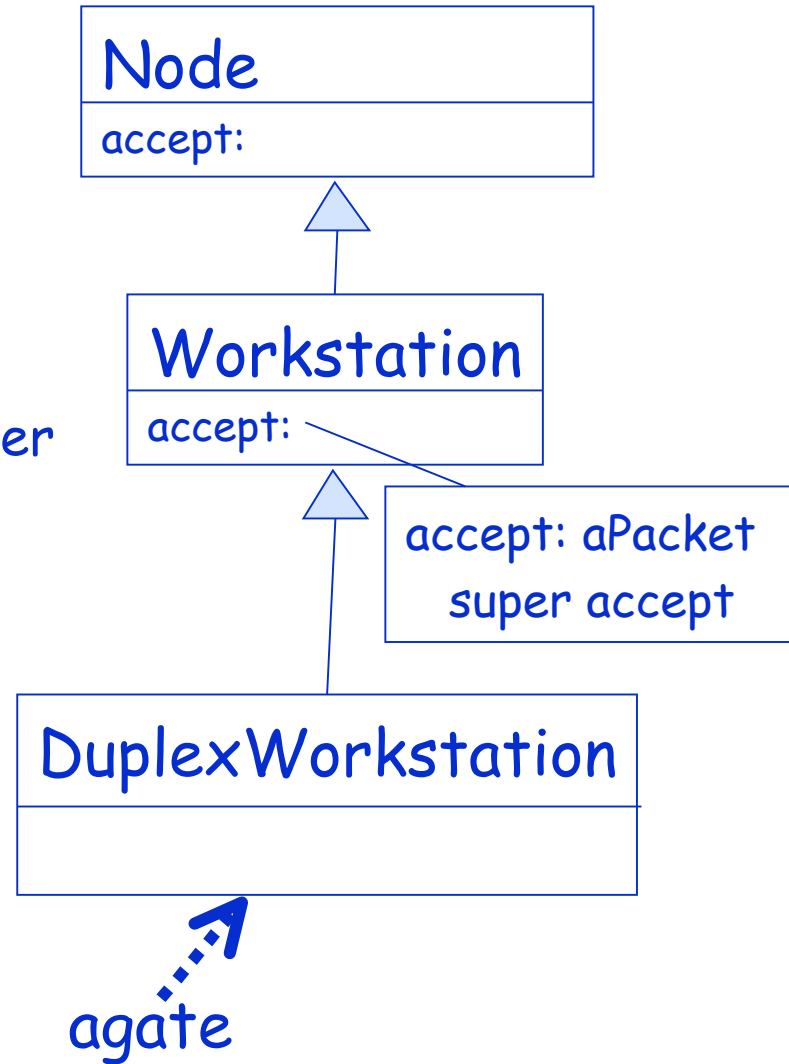


Why super is ****not**** the superclass of the receiver class?

- Let us suppose the **WRONG** hypothesis: "The semantics of super is to start the lookup of a method in the superclass of the receiver class"
 - agate accept: aPacket
- agate is an instance of DuplexWorkstation.
 - accept: is looked up in the class DuplexWorkstation
- accept: is not defined in DuplexWorkstation, so the lookup continues in Workstation

Yes Why?

- accept: is defined in Workstation, so the lookup stops, and the method is executed
- Workstation>>accept: does a super accept:
- Our hypothesis: super = start the lookup in the superclass of the receiver class. The superclass of the receiver class is Workstation
- This will result in a loop, therefore the hypothesis is **WRONG**



What you should know

- Inheritance of instance variables is made at class definition time.
- Inheritance of behavior is dynamic
- `self` ****always**** represents the receiver
- Method lookup starts in the class of the receiver.
- `super` represents the receiver but method lookup starts in the superclass of the class using it.
- `Self` is dynamic // `super` is static