## de de

### INTERPRETATION OF OBJECT-ORIENTED LANGUAGES

Éric Tanter PLEIAD Laboratory Computer Science Dept (DCC) University of Chile etanter@dcc.uchile.cl

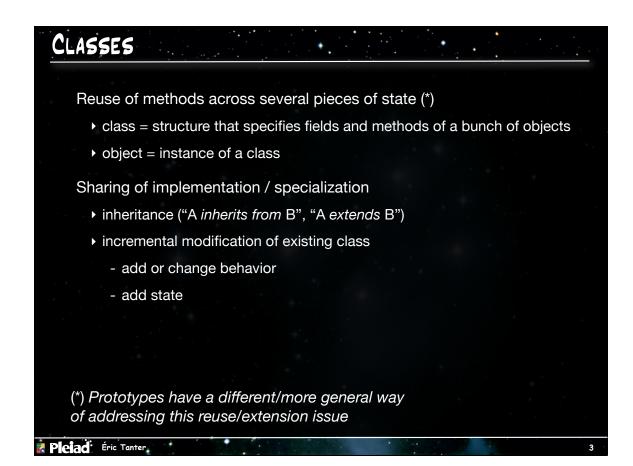


### OBJECT-ORIENTED PROGRAMMING

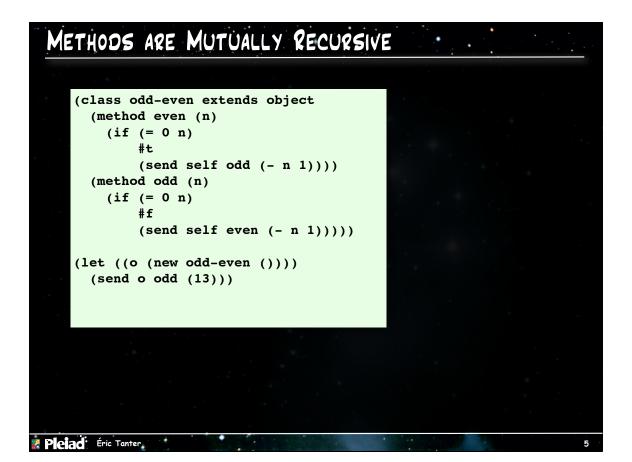
- object = piece of state + behavior (coherent whole)
  - Iocal state: fields
  - methods: behavior that has access to fields
    - calling a method <=> message passing

in higher-order procedural languages with state (Scheme)

- a closure is an object
- fields: free variables
- method: only one! apply!



(class in-node extends object	
(field left)	
(field right)	
(method init (l r)	
(begin (set! left l)	
<pre>(set! right r)))</pre>	
(method sum ()	
(+ (send left sum ())	
(send right sum ())))	
(class leaf extends object	
(field value)	
(method init (v) (set! value v	<b>v</b> ))
(method sum () value))	
	(let ((ol (new in-node
	((new in-node
	((new leaf (3))
	(new leaf (4))))
	(new leaf (5)))))
	(send ol sum ()))



### INHERITANCE

Incremental modification of existing classes

- parent, superclass
- child, subclass: c2 < c1</p>

Single vs. multiple inheritance

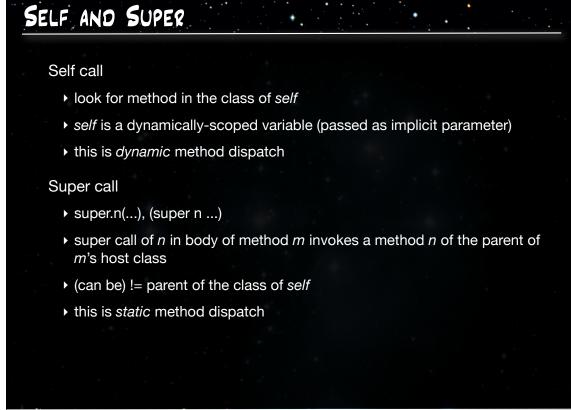
Subclass polymorphism

▶ instances of c2 < c1 can be used anywhere instance of c1 can.

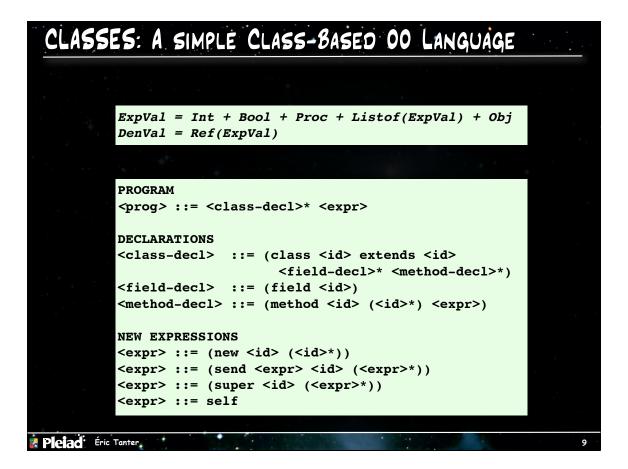
### Redefinitions

- fields: shadowing (lexical scoping)
- methods: overriding

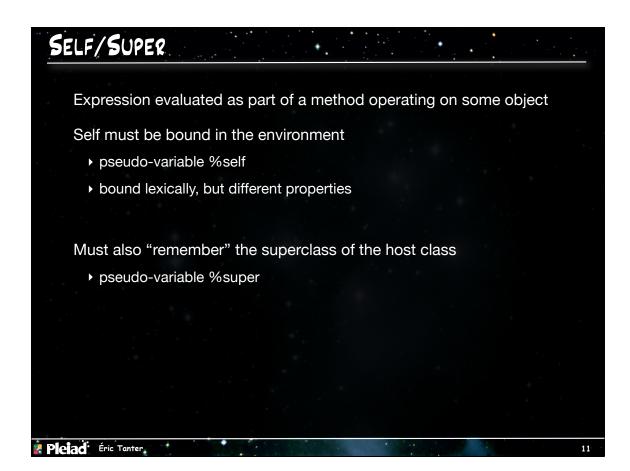
## **TERMINOLOGY: HOST CLASS** Host class of a method • class in which a method is declared Host class of an expression • host class of the method (if any) in which the expression occurs Superclass of a method or expression • parent class of the host class For a given "operation", potentially many methods / host classes.



Pleiad Éric Tanter



# INTERPRETATION PHASES 1. Elaboration of classes all class declarations are processed initialize a global class environment (name -> cls) 2. Interpretation of expressions start with the 'startup expression' (eg. Main.main()) manage environment, environment-passing style deal with new kinds of expressions



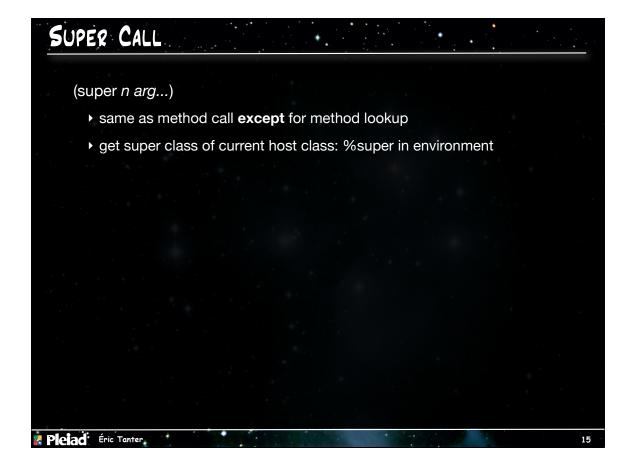




### METHOD CALL

(send obj n arg...)

- evaluate arguments and object expressions
- Iook in object to get class name
- get the class
- Iook up the method (find-method: cname mname -> method)
- apply method



### OBJECT CREATION

(new c arg...)

- evaluate arguments
- using class name *n*, create new empty object
- call initialize method on new object (ignore result)
- ▶ return new initialized object



### OBJECTS

Data structure

- class name
- ► list of (references to) fields

### Creating a new object

- get field list from class
- create new list of fields with refs to illegal values

# DETHODS Like a normal procedure, except that it does not have a saved environment keeps track of the names of the fields it "sees" the name of its superclass Applying a method run body in environment where: formal parameters bound to arguments (by-copy semantics) %self bound to current object %super bound to method's superclass

Pleiad Éric Tanter

# LOOKING UP FIELDS Field shadowing a method "sees" the most recently defined fields \*at its level\* keep position constant if field not redefined field list built from left to right in c1: (x y) [0 1 2 3] in c2: (x y y) [0 1 2 3] in c3: (x y y z) [0 1 2 3] lookup should use *last* position! dedicated lookup, or shadow field names: in c3: (x y%1 y z)

Pleiad Éric Tanter

19

## CLASSES Get information for a class from its name • class environment: define new class, lookup class • classes are static, top-level entities (globally visible) Data structure for classes • superclass name (#f if object) • field names • method environment Class elaboration phase • start with empty class environment • initialize with class object • for each declaration, add a new binding in class environment

21

Pleiad Éric Tanter