

What's missing

The peculiar status of classes (which are both run-time and compile-time things)

Named types with declared subtyping

Recursive types

Run-time type analysis (casting, etc.)

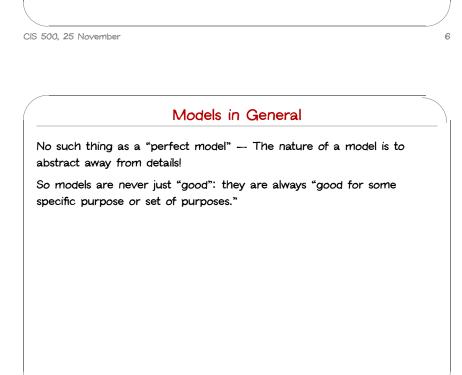
(Lots of other stuff - e.g., ...?)

CIS 500, 25 November

5

Quick Check

How many non-Java-hackers in the room?...



Modeling Java

Models of Java Featherweight Java Lots of different purposes in lots of different kinds of models Purpose: model the "core OO features" and their types and nothing else. & Source-level vs. bytecode level Models of type system vs. models of run-time features (not entirely separate issues) & Models of specific features (exceptions, concurrency, reflection, class loading, ...) Models designed for extension & Models designed for extension g Cls 500, 25 November 2 S 500, 25 November

Featherweight Java

Purpose: model the "core OO features" and their types and nothing else.

Things left out...

♦ Reflection, concurrency, class loading, inner classes, ...

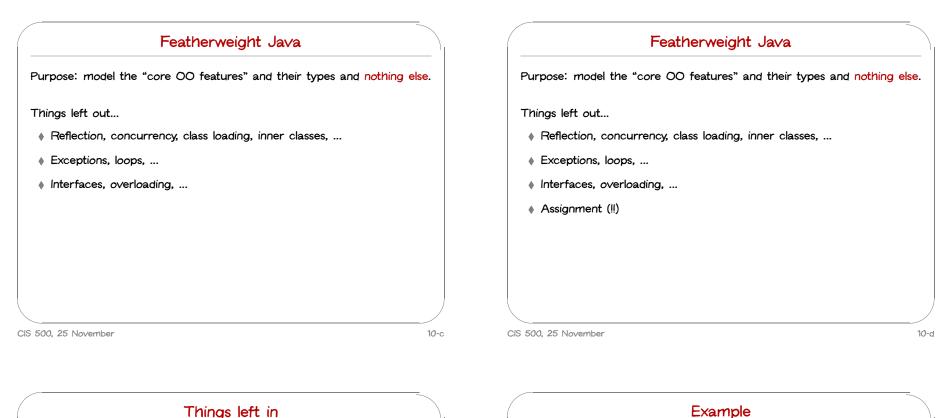
Feath	erwe	iaht	Java
		.9	

Purpose: model the "core OO features" and their types and nothing else.

Things left out...

• Reflection, concurrency, class loading, inner classes, ...

Exceptions, loops, ...



Things left in

- Classes and objects
- Methods and method invocation
- Fields and field access
- Inheritance (including open recursion through this)
- Casting

<pre>class B extends Object { B() { super(); } }</pre>
<pre>class Pair extends Object { Object fst; Object snd;</pre>
<pre>Pair(Object fst, Object snd) { super(); this.fst=fst; this.snd=snd; }</pre>
<pre>Pair setfst(Object newfst) { return new Pair(newfst, this.snd); } }</pre>

class A extends Object { A() { super(); } }

CIS 500, 25 November

Conventions

For syntactic regularity...

- Always include superclass (even when it is Object)
- Always write out constructor (even when trivial)
- Always call super from constructor (even when no arguments are passed)
- Always explicitly name receiver object in method invocation or field access (even when it is this)
- Methods always consist of a single return expression
- Constructors always
 - * Take same number (and types) of parameters as fields of the class
 - Assign constructor parameters to "local fields"
 - Call super constructor to assign remaining fields
 - Do nothing else

CIS 500, 25 November

13

Formalizing FJ

Advantages of Structural Systems

Somewhat simpler, cleaner, and more elegant (no need to always work wrt. a set of "name definitions")

Easier to extend (e.g. with parametric polymorphism)

Caveat: when recursive types are considered, some of this simplicity and elegance slips away...

Nominal type systems

Big dichotomy in the world of programming languages:

- Structural type systems:
 - What matters about a type (for typing, subtyping, etc.) is just its structure.
 - Names are just convenient (but inessential) abbreviations.
- Nominal type systems:
 - Types are always named.
 - Typechecker mostly manipulates names, not structures.
 - Subtyping is declared explicitly by programmer (and checked for consistency by compiler).

CIS 500, 25 November

15

14

Advantages of Nominal Systems

Recursive types fall out easily

Using names everywhere makes typechecking (and subtyping, etc.) easy and efficient

Type names are also useful at run-time (for casting, type testing, reflection, \dots).

Java (like most other mainstream languages) is a nominal system.

CIS 500, 25 November

17



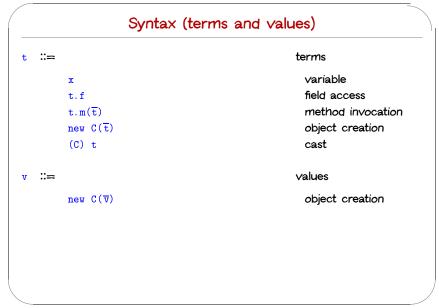
Our decision to omit assignment has a nice side effect...

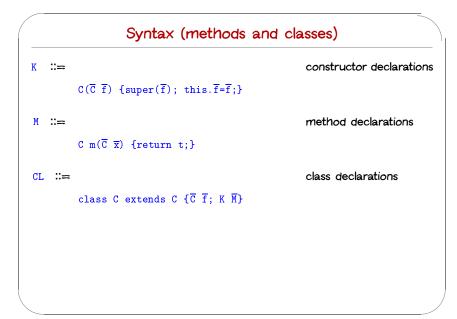
The only ways in which two objects can differ are (1) their classes and (2) the parameters passed to their constructor when they were created.

All this information is available in the new expression that creates an object. So we can identify the created object with the new expression.

Formally: object values have the form $new C(\overline{v})$

CIS 500, 25 November



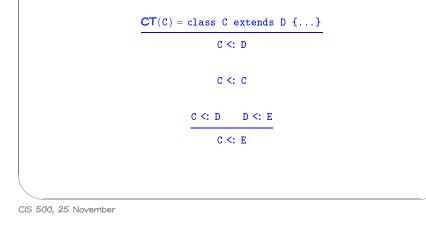


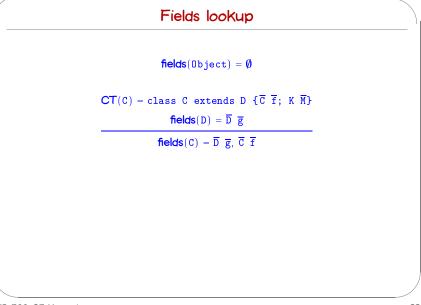
18

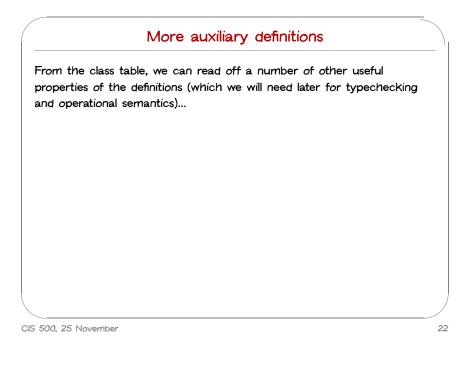
Subtyping

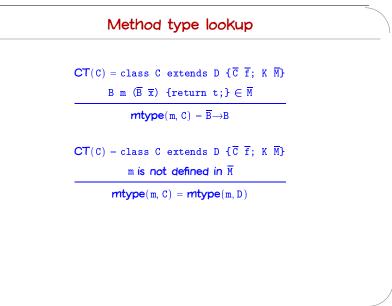
As in Java, subtyping in FJ is declared.

Assume we have a (global, fixed) class table CT mapping class names to definitions.









CIS 500, 25 November

21

CIS 500, 25 November

Method body lookup		Valid method overriding
$\mathbf{CT}(\mathbf{C}) = \mathbf{class} \ \mathbf{C} \ \mathbf{extends} \ \mathbf{D} \ \{\overline{\mathbf{C}} \ \overline{\mathbf{f}}; \ \mathbf{K} \ \overline{\mathbf{M}}\}$		$\textbf{mtype}(m,D) = \overline{D} \rightarrow D_0 \text{ implies } \overline{C} = \overline{D} \text{ and } C_0 = D_0$
B m $(\overline{B} \ \overline{x})$ {return t;} $\in \overline{M}$		$override(m, D, \overline{C} \rightarrow C_0)$
$mbody(\mathtt{m},\mathtt{C}) = (\overline{\mathtt{x}},\mathtt{t})$		
$\mathbf{CT}(\mathbf{C}) = \mathbf{class} \ \mathbf{C} \ \mathbf{extends} \ \mathbf{D} \ \{ \overline{\mathbf{C}} \ \overline{\mathbf{f}}; \ \mathbf{K} \ \overline{\mathbf{M}} \}$		
m is not defined in $\overline{\mathbb{M}}$		
mbody(m, C) = mbody(m, D)		
25 November	25	ClS 500, 25 November

CIS 500, 25 November