CSE 341: Programming Languages

Winter 2005

Lecture 25— Named Types; Class vs. Types; Interfaces

Named Types

In Java/C++/C#/..., types don't look like `[t10 m1:(t11,...), ..., tn0 mn(tn1,...)]`.

Instead they look like `C` where `C` is a class or interface.

But everything we just learned about subtyping still applies!

Yet the only subtyping is (the transitive closure of) declared subtypes (e.g., `class C extends D implements I,J`).

Given *types* `D`, `I`, and `J`, ensure objects produced by *class* `C`'s constructors can have subtypes (more methods, contra/co, etc.)

Named vs. Unnamed

For preventing "message not understood", unnamed ("structural") types worked fine.

But many languages have named ("nominal") types.

Which is better is a tired old argument, but fortunately it has some interesting intellectual points (unlike emacs vs. vi).

First, frame the question more narrowly: Should subtyping be nominal or structural? (Named types don't preclude structural subtyping, e.g. casting between two otherwise-unrelated interfaces.)