CSE 341: Programming Languages

Hal Perkins

Spring 2011

Lecture 25— Extensibility in OO and FP

One-of types and operations

- Given a type with several variants/subtypes and several functions/methods, there's a 2D-grid of code you need:

	Int	Negate	Add	Mult
eval				
toString				
hasZero				

- OO and FP just lay out the code differently!!!
- Which is more convenient depends on what you're doing and how the variants/operations "fit together"
- Often, tools let you view "the other dimension"
- Opinion: Dimensional structure of code is greater than 2–3, so we'll never have exactly what we want in text.

Extensibility

Life gets interesting if need to extend code w/o changing existing code.

- ML makes it easy to write new operations; Java does not
- Java makes it easy to write new variants; ML does not
- In ML the original code must *plan* for extensibility using polymorphism and function arguments
- In Java the original code must *plan* for extensibility using "extra" abstract methods
  - (see "the visitor pattern" on your own)

Unextensibility

Extensibility is not all it's cracked up to be:

- Makes original code more difficult to change later
- Makes code harder to reason about locally (e.g., dynamic dispatch or functions-as-arguments mean you never know what code might execute next)

ML and Java have different defaults, but both let you decide what to make extensible:

- ML: Generally less extensible. Without a type constructor or a function-argument, you limit what might happen (thanks to closed recursion)
- Java: Generally extensible by default. But you can declare methods or classes `final`; arguably under-used.