CSE 341: Programming Languages

Dan Grossman

Spring 2008

Lecture 26— Extensibility in OO and FP

You have grading to do

I am going to distribute course evaluation forms so you may rate the quality of this course. Your participation is voluntary, and you may omit specific items if you wish. To ensure confidentiality, do not write your name on the forms. There is a possibility your handwriting on the yellow written comment sheet will be recognizable; however, I will not see the results of this evaluation until after the quarter is over and you have received your grades. Please be sure to use a No. 2 PENCIL ONLY on the scannable form.

I have chosen *(name)* to distribute and collect the forms. When you are finished, he/she will collect the forms, put them into an envelope and mail them to the Office of Educational Assessment. If there are no questions, I will leave the room and not return until all the questionnaires have been finished and collected. Thank you for your participation.

I'll come back in 20 minutes.

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.