# CSE 341:
# Programming Languages

Dan Grossman

Fall 2004

Lecture 27— 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 at 12:45.*

# Each-of types and operations

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

  |          | Int | Negate | Add | Mult |
  |----------|-----|--------|-----|------|
  | eval     |     |        |     |      |
  | toString |     |        |     |      |
  | hasZero  |     |        |     |      |

- OO and FP 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 we have on a computer, so we'll always have limits in text-based languages.

# 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:
  - For operations, use polymorphism and function arguments. For example, use folds or even abstract the constructors.
  - For types, can use polymorphism, but the lack of subtyping makes it awkward to use the extended types.

- In Java the extended code can *suffer from downcasts*...
  - For types, use interfaces and have new classes implement them.
  - For operations, must downcast to know the operations exist.

- ... or plan for extensibility (e.g., the "visitor pattern")

# 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 not extensible by default. Without a type constructor or a function-argument, you limit what might happen

- Java: Generally extensible by default. But you can declare methods or classes `final`; arguably under-used.

HW6 investigated "unintended extensions" – I think they're usually a bad idea.