Object-disOriented

Dennis Brylow
Marquette University

Programming in the Large

"We argue that structuring a large collection of modules to form a *system* is an essentially distinct and different intellectual activity from that of constructing individual modules. ... Correspondingly, we believe that essentially distinct and different languages should be used for the two activities."

-- DeRemer and Kron, IEEE Transactions on Software Engineering, 1976.

A New Paradigm

- **Simula** had introduced Object-Orientation in the 1960's, but it didn't really catch on until **Smalltalk** in the 1980's.
- OOP achieved mainstream acceptance with **C++**.
- **Java** combined OOP with a strong type system, garbage collection, familiar syntax, and a vast library.

Terminator 3:

Rise of the Virtual Machines

- By late 1990's **Java** had supplanted **Pascal** and **C/C++** as the dominant language in introductory computing courses.
- Some taught the “Pascal subset” of **Java**.
- Others sought to leverage **Java**'s advanced features to rethink what was foundational in teaching software design.

“Objects First”

- Concentrate on top-down design of the system, breaking into logically grouped objects with well-defined interfaces and internal, protected state.
- Inheritance relates objects with similar state or behavior, minimizing code duplication and simplifying maintenance.
- Skip the “*Art of Writing Small Programs Badly*”.

Who is this fringe lunatic?

- BS Computer Science and BS Electrical Engineering from Rose-Hulman
- Ph.D from Purdue (Static Analysis of Interrupt-Driven Software)
- Decade of experience teaching undergraduate computer science: CS180 (5 terms), CS240 (2 terms), CS352 (2 terms), ECE 469 (1 term), COSC 065 (4 terms), COSC 125 (2 terms), COSC 152 (3 terms), COSC 170 (1 term), etc.
- Proponent of bottom-up, systems-oriented coursework on practical software problem solving.

What's Wrong With Java First?

In a first semester, beginners:

- Won't understand objects;
- Won't understand inheritance;
- Can't follow complex asynchronous control flow before they've mastered simple control flow;
- Can't break problems into discrete methods because they don't know what's in a method.

So, What Should Come First?

- Small Language
- Representation
- Functional Decomposition, Algorithmic Thinking
- Fundamental capabilities of machine computation
- Discipline and methodology that will remain valuable even when the dominant language paradigm shifts again.

Tools For the Future

What do we teach that will still be relevant in five years, perhaps in another field altogether?

- Methodical problem solving
- The algorithmic building blocks common to all computer languages
- How to learn about complex technology.

“Everything should be built top-down...

except the first time.”

Programming-in-the-small is tedious, challenging work that we all would like to have skipped in order to move on to the really exciting stuff.

But some mistakes need to be made the hard way in order to learn the really valuable lesson.