CSCI 334:
Principles of Programming Languages

Lecture 17: Object Oriented Programming

Instructor: Dan Barowy

Williams

Topics

"Do it the hard way."

Programming in the large

Dan Ingalls talk

Dan Ingalls on Smalltalk

Topics

"Do it the hard way."

Programming in the large

Dan Ingalls talk

Thursday: more OOP & F# unit testing

Project Timeline

Project checkin: minimally working by May 10

Project checkin: mostly working by May 18

Project done: complete by May 25

Project "presentation": May 25

- 5-10 minute presentation/screen recording
- or a ≥ 1000 word written tutorial (e.g., https://docs.python.org/3/tutorial/introduction.html)

Student question:

"How do I build gumption?"

"[T]here's a school of mechanical thought which says I shouldn't be getting into a complex assembly I don't know anything about. I should have training or leave the job to a specialist. That's a self-serving school of mechanical eliteness I'd like to see wiped out. That was a "specialist" who broke [...] this machine. I've edited manuals written to train specialists for IBM, and what they know when they're done isn't that great. You're at a disadvantage the first time around and it may cost you a little more because of parts you accidentally damage, and it will almost undoubtedly take a lot more time, but the next time around you're way ahead of the specialist. You, with gumption, have learned the assembly the hard way and you've a whole set of good feelings about it that he's unlikely to have."

—Robert Pirsig, Zen and the Art of Motorcycle Maintenance

Jerry: In the old days (before television), being able to add up a long column of numbers without making any mistakes was a valuable skill. People would pay you a living wage to do nothing but add numbers well. Not today.

Theo: Today, it's nice to be able to add small numbers, and larger numbers in a pinch, but the specific mental tricks and habits needed to get the right answer consistently when adding lots of numbers are just not helpful. Not being able to do this does not represent a failure of the intellect, any more than not knowing which fields in your neighborhood have the best rabbit hunting: both were, at one time, failings that would get you laughed at.

Jerry: But, you'd agree that being able to *estimate* the sum of a column of numbers is valuable. I would spend more time learning to do *that* well than working to reduce my error rate in doing exact sums.

Theo: And yet, in schools you find worksheets with 100 addition problems that are supposed to be done *correctly*, with points taken off for errors. What a waste of time.

Irate bystander: Oh, now I get it. You're one of those romantic educational know-nothings who think it's not necessary to learn anything in particular, as long as you learn "critical thinking skills" and have good self esteem. Yuck.

Theo: *No*, and let me make this very clear. No one can learn to think without having something to think about. If you try to teach someone *how* to think in the abstract, you are not going to get anywhere. If you try to make education "easy", by removing the content, you are cheating your students out of the most important thing you have to offer: the chance to do something *hard*. Only by mastering a difficult body of knowledge can a [student] develop into a confident, thinking adult. The point is, it doesn't necessarily have to be the *same* difficult body of knowledge that the [their] parents learned.

And while we're on the topic of romantic educational know-nothings, let me just say that if you think you can improve your students' self esteem by letting them "succeed" at various insipid educational games, you are kidding yourself. [Students] are *much* smarter than that. There is nothing more demoralizing to most children than being put through an educational program they know they can't fail at. Instead of teaching them self esteem, it teaches them that you expect so little of them that you have contrived special extra-stupid lessons for their benefit. Don't think for a *minute* they don't know what's going on.

If you start a lesson off by telling the students "This is going to be easy", you are simultaneously telling them "We had to make this easy because we don't think you're capable of doing anything hard". And when the lesson is over, the only sense of accomplishment they can feel is that they did something easy. So what?

Learning is hard work. If you are not working hard, you are not learning. Period. [Students] love hard work, as long as they see where it's going and why. Instead of killing that energy by giving them something *easy*, we should foster it by giving them something *really hard*. We should tell them it's hard. We should give them the chance to do something meaningful.

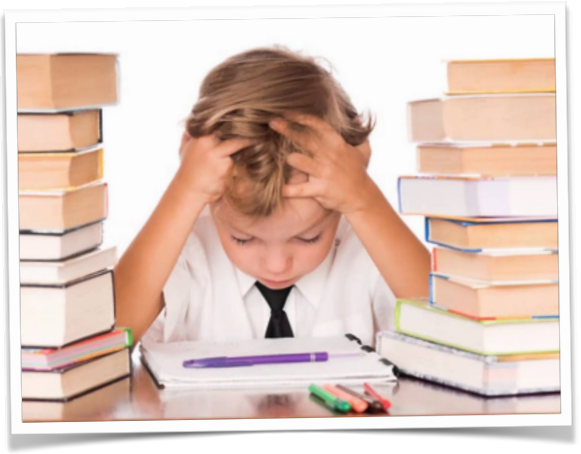
—Theodore Gray, The Beginner's Guide to Mathematica, V4

My advice:

Do things the hard way!

Object-Oriented Programming

Programming in the small



Programming in the large

TensorFlow

Google

Google Search I'm Feeling Lucky

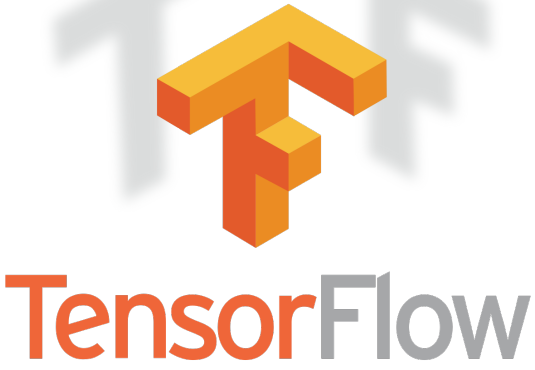
This season, support the local spots you love with reviews and photos on Google

What languages?

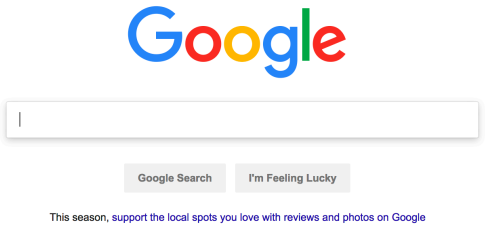
Java

C++

Ruby



C++, Python



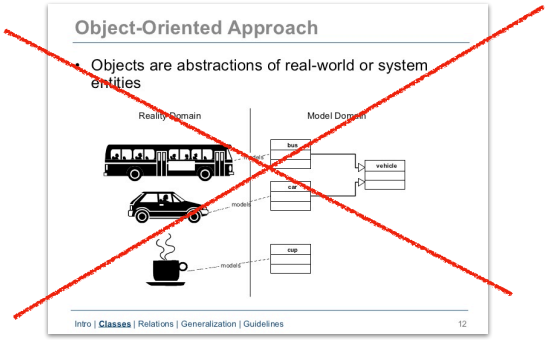
Java

Object-Oriented Programming

- OOP is both a language design philosophy and a way of working (OO design).
- OOP is possibly the most impactful development in the history of programming languages.

What OOP is Not

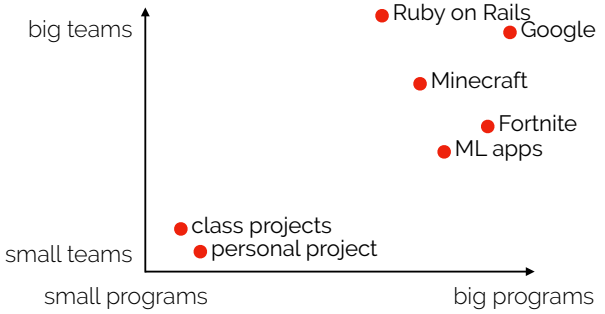
- Many, many instructors introduce OOP as a way of naturally simulating the world.



- This misses the point of OOP entirely!

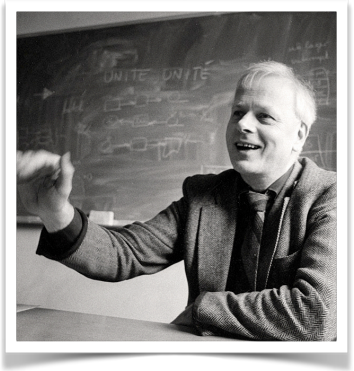
What OOP is

- Object-oriented programming is actually about scalability.
- Scalability in codebase size was the original motivation.
- But OO philosophy also has had a big effect on the scalability of programming teams.



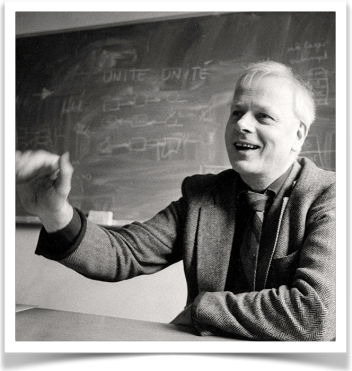
History

- First language recognizable as OO: Simula-67.
- Developed by Kristen Nygaard and others at the Norwegian Computing Center.
- Grew out of frustrations using ALGOL.
- Original plan was to add an "object" library, inspired by C.A.R. Hoare's "record classes".
- It was eventually realized that objects were a fundamentally different way of structuring a program; Simula became its own language.



History

- But Simula-67 was not the most influential OO language.
- That language was…



Smalltalk

Alan Kay
Essentially invented the laptop/tablet ("Dynabook")

Turing Award

Dan Ingalls
Essentially invented object oriented programming

Grace Murray Hopper Award

Adele Goldberg
Essentially invented graphical user interfaces

ACM Software Systems Award

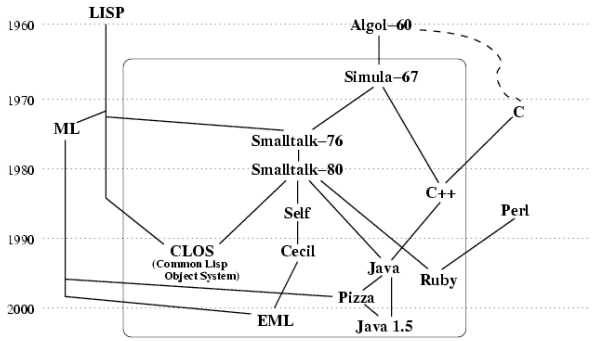
Smalltalk

- First mainstream OO success: Smalltalk
- Developed by Alan Kay, Dan Ingalls, and Adele Goldberg at Xerox PARC and later Apple Computer.
- Used to implement major components of the groundbreaking Xerox Alto computer: OS, compiler, GUI, applications.
- Highly influential. E.g., C++, Java, Ruby, etc.

Smalltalk

And they showed me really three things. But I was so blinded by the first one I didn't even really see the other two. One of the things they showed me was object orienting programming they showed me that but I didn't even see that. The other one they showed me was a networked computer system… they had over a hundred Alto computers all networked using email etc., etc. I didn't even see that. I was so blinded by the first thing they showed me which was the graphical user interface… within you know ten minutes it was obvious to me that all computers would work like this some day.

Smalltalk



OK, really, what is OO?

Object-oriented programming is composed primarily of four key language features:

1. Abstraction
2. Dynamic dispatch
3. Subtyping
4. Inheritance

Purpose: polymorphism at scale

OK, really, what is OO?

Object-oriented programming is composed primarily of four key language features:

1. Abstraction
2. **Dynamic dispatch**
3. Subtyping
4. Inheritance

In my mind, this is OO's killer feature.

Purpose: polymorphism at scale

Recap & Next Class

This lecture:

OOP

Next lecture:

Dan Ingalls' talk on Smalltalk

How does OO work?