Object-Oriented Design

Steps in building an OO program:

- identify the major data abstractions in the application ("find the objects")
- identify the major operations on the data abstractions ("find the interfaces")
- identify commonalities among the data abstractions, identify major implementations of the data abstractions, and organize the abstractions into inheritance hierarchies
- implement the design

An iterative process

Design for the long term:
future applications build upon work done for earlier applications

To inherit or not to inherit

Use inheritance when:

- one ADT is **a special kind of** another ADT
  - "A is a B"
  - interface of subclass is a superset of interface of superclass
- code of one ADT can be reused in large part for code in another ADT
- helps organize & simplify the code

Don't use inheritance when:

- one ADT is more logically **a component of** the other
  - "A has a B"
  - interfaces are different
  - use **aggregation** instead, via instance variables
- only a few routines can be reused without change

If in doubt, don't inherit

Organizing for common interfaces rather than implementation usually better in the long run

Issues with inheritance

Sometimes, interfaces are right, but implementation isn't
⇒ refactor superclass into abstract and concrete classes, inherit from abstract class

Sometimes, could inherit from several things
⇒ use multiple inheritance, if language allows it
(Smalltalk doesn't, C++ does, Java does for interfaces)

A case study: the Smalltalk collection classes

Goal: organize various collection ADTs into useful inheritance hierarchy

- maximize internal code reuse
- maximize uniformity of interface to support more polymorphism in client code

Kinds of collection ADTs:

- Array
- String
- LinkedList
- OrderedCollection
- Dictionary
- Set
- Bag
- SortedCollection
- ...

How to organize them into a hierarchy?
How to define their interfaces in a common way?