Inheritance and Interfaces

- Inheritance models an "is-a" relationship
  - A dog is a mammal, an ArrayList is a List, a square is a shape, ...
- Write general programs to understand the abstraction, advantages?

```java
void doShape(Shape s) {
    System.out.println(s.area());
    System.out.println(s.perimeter());
    s.expand(2.0);
}
```

- But a dog is also a quadruped, how can we deal with this?

Single inheritance in Java

- A class can extend only one class in Java
  - All classes extend Object — it’s the root of the inheritance hierarchy tree
  - Can extend something else (which extends Object), why?
- Why do we use inheritance in designing programs/systems?
  - Facilitate code-reuse (what does that mean?)
  - Ability to specialize and change behavior
    - If I could change how method foo() works, bar() is ok
  - Design methods to call ours, even before we implement
    - Hollywood principle: don’t call us, ...

Multiple Interfaces

- Classes (and interfaces) can implement multiple interfaces
  - A dog is a mammal, a quadruped, a pet
  - How come canine is different?
  - What behavior do quadrupeds have? Pets have?
- An interface specifies the name (and signature) of methods
  - No implementation, no state/fields
  - Yes for constants
- In this course, by convention, we’ll always use interfaces
  - Emphasize design before implementation
  - Use abstract/default classes for code reuse, state

Alan Kay: Turing Award 2004

"Simple things should be simple. Complex things should be possible".
"The best way to predict the future is to invent it"

Kay, because of his experience with children, his love of education, his diverse interests, and his genius, recognized that users can and should interact with a computer in different ways and should not be limited to only text. He was among the first to represent objects in a computer as pictures — a metaphor that he further extended by developing the concept of object orientation. He is, clearly, one of the fathers of the modern PC.

http://ei.cs.vt.edu/~history/GASCH.KAY.HTML
MVC: Model, View, Controller

- A model is the state and brains of a system
  - In a game it's all the pieces and where they are
  - In a spreadsheet it's the data and the formulae
- The view is how we look at the model
  - Spreadsheet has graphs, charts, cells, text, ...
  - Game has board, number of opponents, hit-points, ...
- When the model changes, the views reflect the changes
  - The model tells the views how/if it has changed
  - Model sends information to views OR
  - View asks model for information

MVC: interfaces and inheritance

- A model might have multiple views
  - Tell all the views "I've changed"
  - Who manages the views? This requires state: store views
  - Why can't we keep this state in an interface?
- See IModel and AbstractModel
  - One specifies behavior, the other provides default
  - Don’t rewrite code if we don't have to, maintaining views will be the same for all models
- See IView and SimpleView
  - No default/shared view state/behavior: text and GUI

Does SimpleViewer know Anagrams?

- What does the SimpleViewer know about its model?
  - If we look at code, is there any application-specific logic?
  - What if we wanted to play a game, start a new game?
- Control in MVC with SimpleViewer and IModel
  - Loading a file calls initialize()
  - Entering text calls process()
  - Model calls view with messages, errors, and complete update
- This isn't complete general, but it's pretty generic
  - For this input, here's the output

Comparable and Comparator

- Both are interfaces, there is no default implementation
  - Contrast with .equals(), default implementation?
  - Contrast with .toString(), default?
- Where do we define a Comparator?
  - In its own .java file, nothing wrong with that
  - Private, used for implementation and not public behavior
    - Use a nested class, then decide on static or non-static
    - Non-static is part of an object, access inner fields
- How do we use the Comparator?
  - Sort, Sets, Maps (in the future)