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?

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 often use interfaces
  - Emphasize design before implementation
  - Use abstract/default classes for code reuse, state

Inheritance Example

```java
public class UtilRecord extends Record {
  // assumes Record’s fields are protected, not private

  private String kind;

  public UtilRecord() {
    this("", ",", 0, 0, ",", ",");
  }

  public UtilRecord(String a, String f, int s, int o, String e, String k) {
    super(a, f, s, o, e);
    kind = k;
  }

  /* etc */
```

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, ...
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)
- Does Hashing (future topic) have similar problems?

Comparable Example

```java
public class Record implements Comparable {
    public Record(String a, String f, int s, int o, String e) {
        ...  
    }
    public Record(Record rec) {
        ...  
    }
    int compareTo(Record r) {
        // comparison code goes here
        // return n to neg # if <, 0 if ==, and pos # if >
        return n  
    }
}
```

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
  - Spread sheet 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