Interfaces Day

* Review concepts of abstract class (was the last thing we did)
  + Remember an abstract class has one or more methods that are marked as ABSTRACT, which means those methods have no bodies. Instead, a method marked as abstract must be overridden in all derived classes.
  + This is used when a concept represented by a class is so general, that we can't write code for that method.
  + Review Pet class and speak.
  + Shape class and getArea.
  + Remember you can never instantiate (create a new object) of an abstract class, because how would you call the abstract methods.
* Today's topic is interfaces.
  + Interfaces are very similar to abstract classes, and they can be used for a lot of the same kinds of things.
  + The difference is that defining an abstract class allows you to do three things:
    - define instance variables
    - define methods (with bodies)
    - define abstract methods (no bodies)
  + An interface is like a class, but only allows you to define abstract methods (no instance variables or methods with code).
  + So what is this good for?
    - Good when you are defining a concept that is completely specified by its behavior (methods) rather than its variables.
    - Also used when you want to specify common behaviors for objects where the objects in question might not be related to each other through an IS-A hierarchy.
* Reintroduce Dog/Cat/Pet/Showdog hierarchy
* Introduce Shape hierarchy. with Circle/Square.
* Suppose now we want to write code that will draw any kind of pet or any kind of shape on a simple canvas.
* Take a look at the DrawingDemo class.
* Notice how we make 5 different objects. Now all of those objects could be drawn on a canvas.
  + But they all would be drawn using different code, right?
  + So imagine I wrote a draw() function inside each of those five classes (one each for dogs/cats/showdogs/square/circle).
  + I could then simply call draw() on each object.
  + But what if there were many many more objects to draw? I couldn't type out a billion draw commands.
* Suppose we want to add these objects into an arraylist and then write a function that will take the arraylist, loop over all the objects, and draw each one?
* How could I create this arraylist? It would have to be an arraylist of what datatype?
  + (Object), because that's the only unifying characteristic.
* public interface Drawable {  
   public void drawOn(SimpleCanvas canvas, int locationX, int locationY);  
  }
* Go over Drawable interface
* What should we make Drawable? Dog/Cat/ShowDog or Pet. Shape or Square/Circle.
* Add Drawable to these classes.
* public void drawOn(SimpleCanvas canvas, int locationX, int locationY) {  
   canvas.drawImage(locationX, locationY, "dog.png");  
  }
* Implement Drawable.
* Now update diagram on board. Show classes that derive from Drawable.
* Note how Dog is now a Pet and a Drawable (as well as an Object).
* So we can make an arraylist of Drawables and put anything drawable in it.
* Call the function (change function to accept arraylist of drawables).
* Show Collections class. ...note sort functions.
* Show Comparable interface.
* Show SortingDemo.
* Make Fraction implement compareTo.
* Make Pet implement Compareto.