Reviewing Class Design

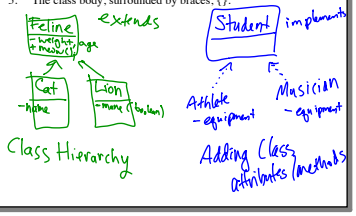
You've seen classes defined in the following way:

```
class MyClass {
    // field, constructor, and
    // method declarations
}
```

class declaration *class body*

Class declarations can include these components, in order:

1. Modifiers such as *public*, *private*, etc.
2. Class name (first letter capitalized by convention)
3. Class's parent (superclass), if any, with the keyword *extends*.
   A) Classes can only *extend* (subclass) **one** parent.
4. Comma-separated list of interfaces with keyword *implements*.
   A) Classes can *implement* **more than one** interface.
5. The class body, surrounded by braces, {}.



Types of variables:

- fields - Member variables in a class
- local variables - Variables in a method or block of code // limited scope
- parameters - Variables in method declarations

Access Modifiers

Controls what classes have access to a member field

`public` and `private`

- `public` modifier—field is accessible from all classes.
- `private` modifier— field is accessible only within its own class.
- "In the spirit of encapsulation," we make fields private (can only be *directly* accessed from the class they are in using the "this" keyword)
- To access *fields* from other classes (*indirectly)* - add public methods that obtain the field values for us (called accessors and mutators, setters and getters)

This - a Java keyword that allows you to refer to the implicit parameter inside a class.

Within an instance method or a constructor, `this` is a reference to the *current object* — the object whose method or constructor is being called

Fields and methods can be declared *static* (this is true in C++, too).

If a field is static, there is only one copy for the entire class, rather than one copy for each instance of the class. (In fact, there is a copy of the field even if there are *no* instances of the class.)

Example class; a List is an ordered collection of items of any type:

```
class List {
   // fields
   private Object [] items;
   // store the items in an array
   private int numItems;
   // the current # of items in the list

   // methods
   // constructor function
   public List() {
     items = new Object[10];
     numItems = 0;
   }   // AddToEnd: add a given item to the end
   //of the list public void AddToEnd(Object ob)
   { ... } }
```

For example, we could add the following field to the List class:

```
static int numLists = 0;
```

And the following statement to the List constructor:

```
numLists++;
```

Now every time a new List object is created, the numLists variable is incremented; so it maintains a count of the total number of Lists created during program execution. Every instance of a List could access this variable (could both read it and write into it), and they would all be accessing the *same* variable, not their own individual copies.