[0631] EXTRA: Favor Composition over Inheritance

Over the years, Inheritance (and its side-kick polymorphism) has lost favor with many real-world programmers. Here is a list of some of the problems they have found with Inheritance:

- Changing a base class' implementation often will break subclasses in mature inheritance hierarchies.
- Inheritance "breaks" encapsulation - sub-classes are allowed to reach directly into the protected members of a superclass.
- Inheritance is defined at compile time and cannot be changed/adjusted at run-time.
- Universally agreed upon inheritance hierarchies don't exist in the real-world.
- Inheritance hierarchies do not map well onto database data.

http://www.mikevalenty.com/inheritance-is-evil-the-story-of-the-epic-fail-of-dataannotationsmodelbinder/ (http://www.mikevalenty.com/inheritance-is-evil-the-story-of-the-epic-fail-of-dataannotationsmodelbinder/)

That said if you are modeling something that is truly, inescapably an "is-a" relationship AND you are sure you fully understand all aspects of those relationships, then you can consider designing an Inheritance of classes. While those kinds of relationships can happen when creating a bottom-up library of classes, it is relatively rare in application programming.

Composition

What is much more common are "has-a" relationships. A data field has a validator. A POJO has an iterator. A collection has a serialization method.

"has-a" relationships are created via a technique called "Composition." Composition is a fancy word for "has a member variable that contains an object that implements a specific interface."

(We will get to interfaces soon. For now, think of an interface as a pure abstract class - one where all the methods are defined via method signatures, but none of them are implemented.)

There a very similar kind of class relationship called **Aggregation.** Aggregation implies a relationship where the child can exist independently of the parent. On the other hand, with Composition, the child cannot exist without the parent. Both Aggregation and Composition are also considered to be different types of **containment**.

Here are some key advantages of composition:

- The public interface of each class is respected. Encapsulation is preserved.
- Composition relationships are established at run-time and can be dynamically changed/updated.
- You can add more than one composition class to your object.

- Composition encourages programming to interfaces instead of concrete types.
- You can control what parts of the superclass are exposed via delegation.