[0620] Inheritance Syntax in C#

To have a subclass inherit from a superclass, add the superclass to end of the subclass' declaration using a colon character.

For example:

```
class Eagle : Bird
{
    // Eagle-specific stuff goes here.  Non-private "Bird"-stuff already included automatically.
}
```

Constructing Objects with Superclasses

Constructing an object using an inherited class needs to be done carefully. The most important thing to remember is that **objects must be constructed starting with the top-most base class and then progressing down through any subclasses and then finally to your class**. In order for this to happen correctly, every subclass constructor needs to "chain" to its superclass' constructor using the colon operator and the "base" keyword. That causes all of the superclass "stuff" to be initialized correctly BEFORE the code in your constructor executes. So the constructor for the Eagle class might look like this:

```
Class Eagle : Bird
{
   private int numTalons;

   public Eagle(string name, int numTalons = 2) : base(name)
   {
      this.numTalons = numTalons;
   }
}
```

": base(name)" causes Bird's constructor - Bird(name) - to be called before any of the code in Eagle's constructor runs (assuming that the Bird class tracks the bird's name.)

The protected Access Modifier

You can use the "protected" access modifier in place of "private" if you want something in your class to be accessible to subclasses but NOT to the outside world. The book thinks this is a good thing. I don't. Use getters and/or setters instead. Lord only knows what crazy things subclasses might do to your member variables.

Sealed Classes

Speaking of cracking down on subclasses, the sealed keyword is the ultimate weapon in that regard. If a class has the "sealed" keyword in its declaration, it cannot be subclassed. In theory, unless you specifically designed a class to behave as a base class, it should be sealed. (In reality, most programmers don't bother - but they should.)

Nested Classes

You can declare classes inside of other classes. Don't. (Well... Don't unless you have a really good reason to. It usually just adds unneeded complexity to the code.)