[0561] Automatic Properties

When your class has a large number of fields, creating Properties for all of them can be a pain. Each Property's get part is just "{ return variable; }" and the set part is just "{ variable = value; }" To help with this, C# has made it very easy to create such Properties, which they call "Automatic Properties" - you just add "get; set;" to your property code and C# handles all the rest! It even creates the "backing variable" (which it hides from you.) Awesome, right?

Well... kinda.

The whole point of encapsulation is to NOT do what fully Automatic Properties do. You do NOT want to just open up a field inside your class to arbitrary modification. At the very least, you want to favor immutability over mutability right? **[0331] Why Immutability is A Good Thing (https://lwtech.instructure.com/courses/1841516/pages/0331-why-immutability-is-a-good-thing)**

In addition, there is very little different between a fully Automatic Property and a public member variable (which, hopefully, we all agree is a bad thing). The only difference is that if, later, you decide that you need to protect you public member variable with some code, you would break any other programs that were relying on it. (I would argue that you need to think hard NOW about protecting that field.)

That said, if you need to quickly create an immutable property that has a default Getter and no public Setter (and you don't need to see the underlying backing variable), Automatic Properties can save you some keystrokes. Here's what a realistic version of our JellyBean class looks like with automatic properties:

```
class JellyBean
{
   public Color BeanColor { get; private set; }
   public int Size { get; private set; }

   public JellyBean(Color beanColor = Color.Black, int size = 10)
   {
      this.BeanColor = beanColor;
      this.Size = size;
   }
}
```

We need to use "private set" in both properties so that they can be set by our constructor. We are left with a very short, compact class that has a lot going on "in the background." This is a ***classic*** C# trade-off!

C# makes it very easy to create compact code as long as you understand what goes on behind the scenes!