[0560] .NET Properties

.NET Properties let you create and use encapsulated fields without defining or calling Getters and Setters.

*Note: It is important to understand that these are **.NET** Properties, not **C#** Properties. That means that these Properties can be used in a cross-language manner and, in fact, that is one of their key benefits. In many ways, .NET Properties seem simpler than the encapsulated fields we just studied, but that simplicity comes at the cost of some flexibility as we will see. In theory, you can mix elements of Properties with elements of encapsulated fields, but in practice, typically, you'll use one or the other, not both.*

To create a Property in a C# class, first create a private field in that class that will contain the property's value. By convention, those fields should be named using camelCase and the first letter of the field should be lower case. Next create the property itself by creating a public version of that same field however name it using PascalCase (i.e., starting with an uppercase letter) and follow the name with a couple of curly braces for the encapsulation code.

Inside those curly braces, you can add "get" code and/or "set" code that executes when the property is read or written to.

For example, using C# Properties, our JellyBean example now becomes:

```
class JellyBean
{
   public const int maxSize = 1000;
   private Color beanColor;
   private int size;

   // Constructor with optional parameters
   public JellyBean(Color beanColor = Color.Black, int size = 10)
   {
      this.beanColor = beanColor;
      this.size = size;
   }

   // Define the BeanColor property with a single getter
   public Color BeanColor
   {
      get { return beanColor; }
   }

   // Define the Size property with both a getter and a setter
   public int Size
   {
      get { return size; }
      set
      {
         if (size > 0 && size < maxSize)
```

```
            size = value;
        }
    }
}
```

And now, people can use our POJO just like it was full of public member variables - i.e., just use the normal dot operator to get and set the variable values (i.e., jb.Size, jb.BeanColor). Just remember that the protective code that validates the input (i.e., the castle guards) will still run ensuring that our data remains uncorrupted.

Notes:

- **The name of the property must be *different* from the name of the "backing" member variable.** There are two coding standards for ensuring that happens: 1.) Use PascalCase for the property name (i.e., capitalize its first letter) while camelCasing the backing member variable (like I did above). 2.) Append an underscore to the backing member variable. In the code example above, that would mean changing "size" to "_size".
- **You can create immutable properties by omitting the set section.** Just like with Getters and Setters.
- **The contextual keyword "value" represents the value that the property is getting set to**, usually via an assignment statement. For the statement "jb.Size = 5;" **value** would be set to 5 before the code in the set section of the property was called.
- Static properties are possible (but not very useful).