[0520] Static == One

When it comes to C# (and Java) programming, "static" means "one." When static is added in front of a class, method or member variable, it changes it into something similar to a global variable - there's only one and it must be shared by everything that tries to use it.

Static Classes

If you declare a class to be static, C# automatically creates one object of that class when the program starts. This is the only kind of objects that aren't created via the **new** operator. In fact, trying to use **new** with a static class results in a compiler error.

When you use a resource from a static class, you simply add the name of the class to the front of the resource, so for instance, Console is a static class that all console programs have access to and to use any of its methods you must add "Console." to the front of the method name - for example, Console.WriteLine("Cling");

Often, static classes are "utility classes" meaning that they have a collection of methods that only work on the parameters you give them. They don't save any "state" in their member variables - typically because they don't have any member variables.

Static Member Variables

Static member variables are... strange. At the very least, they are tricky to use correctly because they are essentially global variables. In my experience, the only common usage for static member variables is for constants that never change. In C#, when you add "const" to the front of a variable's declaration, that means it can only be initialized once and, it is automatically made static which saves some memory. Here is how you create a (implicitly static) constant:

```
class JellyBean
{
   private const int maxSize = 1000;

   private Color beanColor;
   private int size;

   public Color GetBeanColor()
   {
      return beanColor;
   }

   public int GetSize()
   {
      return size;
   }

   public void SetSize(int size)
   {
```

```
      if (size > 0 && size < maxSize)
         this.size = size;
   }

}
```