[0510] Constructors

All Objects are created by the **new** operator. When you use the new operator in your program, it allocates memory for the new object from the Heap and then initializes that memory before returning the initialized Object to your program. Constructors give you the ability to change how that initialization process works. **Constructors are all about object initialization.**

You might notice that what follows the new operator *looks* like a call to the class' constructor. Technically, it isn't - but you can pretend that it is if that helps you understand the process better.

At the very least, when you create an object with new, you use the Class' Default Constructor to create the object. The Default Constructor doesn't take any parameters and just sets all your member variables to their "zero/default" state. Generally, that's rarely what programmers want and so frequently they will define their own default constructor overriding C#'s version.

These days, thanks to C#'s Optional Parameter feature, POJOs usually have a single constructor which can be passed any number of initial data values. The constructor's job is to validate those initial values and then store them in the appropriate member variable(s) AND NOTHING ELSE!

In theory, a constructor can do anything it wants - it could write stuff to the console, it could save stuff in a file, it could send email, you name it. ALL OF THAT IS A BAD IDEA! **Constructors are for initializing member variables - period.** Say that several times until it is seared into your brain.

There are a couple of reasons that constructors should only be used for initializing member variables:

- Constructors should (almost) never fail. If they do fail, it should only be because the computer is very low on memory.
- Constructors are limited in the ways that they can communicate errors back to the program that called them (realistically, all they can do is throw an exception which most calling programs will not be expecting).
- Constructors are generally expected to execute fairly quickly. Programmers rarely expect constructors to slow down their program.
- Constructors are often called frequently. If an array of objects is created, the constructor for that object will be called for each element in the array.
- Constructors sometimes need to execute in low memory conditions. The "thinner" your constructor is, the more objects you will be able to create when memory is tight.

Constructor Chaining

Before there was C#'s concept of optional parameters, programmers would often create what is called a "Constructor Chain." The book has several examples. Constructor chains allowed you to use constructors with 1, 2, 3, or more parameters depending on the needs of the class. Missing parameters would be filled in with default values (which is exactly what optional parameters do!). While you might encounter constructor chains in legacy code, it is now easier to create a constructor that uses named, optional parameters. Here's an example:

```
class JellyBean
{
   private Color beanColor;
   private int size;

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

   // All the Getters and Setters go here...

}
```

What could be simpler? If the constructor is called without a beanColor value, black is used automatically. If the constructor is called without a size value, 10 is used. Combined with the named parameters feature, this is a very convenient style of constructor to use no matter how many fields your POJO has.