**Notes based on Skillsoft course**

**“General Coding Best Practices”**

**Course completed July 29, 2011**

**Code Conventions**

* Code s/b as self-documenting as possible (but will still have to be internally documented via code comments).
* Use white-space liberally as long as it helps readability.
* Limit lines to less than 80 chars.
* Make use of the first two of the three standard naming conventions (***Pascal***, ie DataSet; ***Camel***, ie numberOfDays, ***Hungarian***, ie strFirstName or intCount). Note Hungarian is argued to be outdated.
* Avoid variable names that differ only in case.
* Avoid abbreviations.
* Class names should use a noun and should be in Pascal case.
* Method names should be in Camel case.
* Comments may summarize the code (like at the top of a method), describe the code’s intent (indicate the problem to be solved, the WHY not the HOW, and/or prepare reader for what is to come), or provided information that the self-documenting code can’t express.
* Comments used to explain complicated code should be unnecessary, rather you should try to improve the code. If code is confusing, then comment it.
* Tools, like JavaDoc, exist to generate comments… What is the .net equivalent?

**The Structure of Programs**

* A literal is a notation for representing a fixed value in source code. Literals are often used to initialize variables, ie int a = 1 (1 is a literal), or string s = “abc” (abc is a literal);
* Literal values s/b replaced with named constants. (from above, don’t use the “1” use the variable “a” but make it a “const”).
* Objects don’t contain values – they point to values. So if you use ‘== ‘ to compare two objects you are comparing the pointers instead of the values. Objects in C# have a “.Equals” method. Given obj x, and obj y where x.Equals(y) returns the same value as y.Equals(x).
* Move the contents of complex if statements to separate methods. For example If(x&&5 > 444 || sdfsdf < 4 && 4>uuu) then, s/b if(Method) then, where Method returns the Boolean value of “x&&5 > 444 || sdfsdf < 4 && 4>uuu”.
* When writing code that must execute in sequence be sure to name methods intuitively, and use parameters and return values to indicate dependencies.
* Float, and Double types were designed for scientific calculations and not for monetary calculations. In C# use the Decimal type for monetary calculations. Or you could use Int type by first multiplying the values by 100 to include the digits to the right of the decimal point.
* Declare, and assign to variables as close as possible to where they are used. It is bad practice (left over from the old C days) to declare all variables at the top of a method.
* Favor local variables over global ones.
* Code logic depends on three constructs – decisions (*if, case*), loops(*while, for, foreach*), and sequence (*the order, a happens before b*).
* In conditional statements (if, case) the most likely (aka nominal) condition should come first.
* Three general types of loops. 1. Colletion controlled (foreach, works on enumerations), 2. Condition controlled (While), 3. Count controlled (For).
* Never nest loops more than the maximum of three levels.
* Avoid arbitrary loop breaks via goto, return, break or continue. If you must do this then comment it carefully.

**Methods**

* A method is a section of code that performs a single procedure for a single purpose. That is to say, a *good* method is cohesive, meaning it will perform a single, clearly defined function. Methods that have multiple purpose s/b separated into distinct functionally cohesive methods.
* Two types of methods, 1.Functions return values, 2. Procedures don’t return values. C# just has methods; the return type of VOID makes a method a procedure.
* If global variables are used within a method, then given the same input parameter values the method could return different results, and that is poor coding. A method given the same input parameter values should always return the same values; this is known as being referentially transparent.
* Invariant (computer science), an Expression whose value doesn't change during program execution. In Java, Class Invariants are methods which check the validity of an object's state (its data). The idea is to define validation methods for fields, and to perform these validations whenever the fields change. Generally speaking, class invariants are constraints on the objects in a class.
* Five guidelines to follow when creating a method:
  1. Identify the method’s purpose.
  2. Name the method with a self-documenting name. If it is hard to find a name that describes the purpose of the method then likely the method is not well-defined with a single purpose. You should be able to read the methods name, without reading its code and know what single function it performs.
  3. Reduce side effects: avoid modifying global variables, and avoid hidden dependencies (good encapsulation will avoid this – encapsulated means it modular and self-contained).
  4. Control access to the method (private, public, protected).
  5. Devise a test plan for the method before you write the method, this will help you know exactly the purpose of the method before you code it. This way you can test methods as they are developed.
* C# 4.0 allows methods to have Optional Parameters. Arguably, the advantages of optional parameters are outweighed by the disadvantages. It’s pretty much a form of bad cohesion known as logical cohesion, in which the method needs to handle different situations and perform several tasks based on the value or existence of the optional parameters. This can make the code slower, and obviously harder to read. Microsoft has allowed this in large part due to its plan of co-evolution of C# and VB.NET since VB.NET has had this feature for a while. The better option is to use method overloading.
* Best practice says to distinguish between your input parameters, your working variables, and return (output) parameters. For example don’t take an input variable and work on it, and then return it with its new value – this type of logic is harder to follow. Also only include parameter that are used (hence on optional parameters, and on logical cohesion).
* An exception is code used to handle errors and/or unexpected events.