[0062] On Comments, Indicators Names, and Readable Code

Let me make a bold, controversial statement right up front here:

Comments are rarely needed in well-written programs!

Let's explore that for a second. There are several things that need to be understood about commenting:

- The audience for comments is the set of programmers that will look at your code. This includes other team members, the Maintenance programmers in the Operations team, future team members, and - maybe most importantly - *a future version of yourself!*

- It is safe to assume that ***the people reading your code know how to program***. That means that comments that point out basic features of the language you are using are... useless! Comments that indicate - for example - that this is the start of a loop are just annoying to experienced programmers and should not be added. (And with today's modern IDEs, comments that indicate where a loop ends are also unnecessary.)

- On the surface, big huge blocks of comments explaining how a method or class is implemented seem really helpful. The problem is that, over time, those big blocks of comments will become out-dated and ***out-dated, misleading comments are much worse than no comments at all.*** A key reason those comments will become out-dated and misleading is because of the pressure to fix bugs in production code. Again, those fixes are done by people - often not the original programmer - whose hair is "on fire" i.e., under a ton of pressure. Their goal is to get the bug fixed as quickly as possible and back into production. Once they fix the bug, they rarely/never go back and see if there is some big huge block comment that also needs updating. Thus those block comments (often) will become out of date over time.

- Comments are a non-executing "copy" of the actual code. "Non-executing" means that if the comment says that the variable is incremented, but the code says to decrement the variable, ***the code always wins*** and the variable is decremented.

You must keep the following things in mind when writing programs for this class:

1.) You should always strive to make the code itself as readable as possible. Only add comments when something cannot be made self-evident in the code - which should be rare.

2.) You must use useful, understandable identifier names that make your code as readable as possible!
Each time you create a new "thing" in your program code - a new variable, a new method, a new class, etc. - it is VERY important to give it a name that is meaningful, useful and non-ambiguous. Yes, that can take some time (up to 10 whole seconds!), but it is time well spent. For all the reasons listed above, using meaningful identifier names is super-valuable.

3.) I will deduct points from your Programming Assignment grades if you fail to use meaningful, well thought out identifier names! The suggestions below should help you come up with great identifier names. Follow them at all times.

Identifier Refactoring

Traditionally, programmers hate coming up with identifier names. First off, it takes time and imagination to come up with a good name. Second off, the names usually change several times anyways. And thirdly, they are a huge pain to change! Fortunately, modern IDEs have solved 2.5 of those problems through a feature called "refactoring." Refactoring refers to the process of changing your source code in some way that has no impact on the running version of your program. While there are many different ways to refactor code, the simplest and most commonly used is Indicator Renaming. IDEs like Visual Studio are smart enough to know (based on scoping rules) which versions of a variable you are renaming and so they will only change those versions when you refactor them.

Refactoring doesn't eliminate the need for imagination and intelligence when renaming an indicator, however, it does all you to try out as many different names are you want quickly and easily so you can see which one works best.

In Visual Studio, to Refactor an indicator's name, simply click on the indicator, then press CTRL-R twice. Then type in the new name. As you type, all of the instances of the old name will change instantly which is kinda cool.

Indicator Naming Suggestions

Here are some common C# naming conventions that can help you come up with maintainable indicator names:

- C# uses PascalCase for method names, camelCase for variables.
- Only use i, j (and possibly k) as integer index counters for loops. Using them for anything else will confuse later programmers.
- If the indicator is a boolean type, try to give it a name that naturally equates to yes or no. For instance "IsComplete" is a boolean with an obvious purpose. "HasValues" also works. In fact, in general, see if you can make your boolean variables and boolean methods start with "Is" or "Has". If you can, you probably should.
- Try hard to use "positive" boolean indicator names. Instead of "IsNotReady" use "IsReady". Only use "negative" expressions and conditions if you absolutely have to.
- Avoid using "No" at the start of indicators because it could mean either "No" (as in "not yes") or it could be an abbreviation for "number." If you are creating a variable that counts things, use "num" instead of "no".
- Class names should be singular nouns - e.g., "Person", "Card", "Object"
- Enum names should be singular nouns.- e.g., "Color", "Planet"
- In general, Variable names should be singular nouns unless they contain several items - arrays, collections, etc.
- In general, the names of void methods should start with an active verb and not include the word "And" (because, as we will see, methods should only do one thing). Examples: "PrintStats()", "DeleteTree()", "Init()"
- In general, names of methods that return a value should be the name of the value they return. Examples: "Min()", "Max()", "AbsVal()"
  - Exception: Methods that allow access to fields inside an object typically start with "Get". Examples: "GetName()", "GetAddress()"

- In general, do not add abbreviations or "type prefixes" to the front of identifiers.
  - Exception: "I" should be added in front of all Interface names.

When considering the use of these (or any other) code conventions, remember the two "master" rules that override everything:

1.) Try to do what the majority of other programmers do.
2.) Don't do it if it makes the code harder to read or more confusing.