C# coding standards Cheat Sheet

by filoucool via cheatography.com/157372/cs/33343/

Control Flow

Avoid changing a loop variable inside a for loop block.

Try to update loop variables close to where the loop condition is specified.

Always put a default label for switch statements

Avoid conditions with double negatives.

Object Lifecycle

Try to declare variables close to where they are used.

Try to initialize variables at the point of declaration.

Never shadow a name in an outer scope.

Avoid implementing a finalizer.

Avoid empty finalizers.

Always implement IDisposable if a classes uses unmanaged resources or owns disposable objects.

Never call a virtual method unless the object is fully constructed.

Exceptions

Never silently ignore exceptions.

Try to use standard exceptions.

Try to throw the most specific exception possible.

Always log when an exception is thrown. Log verbosely.

Always create verbose error messages.

Always list the explicit exceptions a method or property can throw.

General Guidelines

Always favor simplicity. Write the simplest code that will work. KISS.

Always favor readability. Name your variables and methods as clearly and as descriptively as possible.

Try to keep methods under 25 lines of code (excluding vertical spacing and comments). Break down your code into small functions that are easy to understand. If a method is over 30 lines of code, it should be refactored.

Try to keep classes under 400 lines of code (excluding vertical spacing and comments). If a class is over 500 lines of code, it should be split out into separate classes that each do one thing.

Try to keep your code DRY, extract duplicate code into methods.

General Guidelines (cont)

Try to include design-pattern names such as Bridge, Adapter, or Factory as suffix to class names where appropriate.

Avoid duplicating constants. Separate into a config file.

Always delete unreferenced files.

Try to have lines less than 80 characters. If the line exceeds 160 characters, it should be refactored.

Never have more than two nested calls on the same line.

Never have methods with more than three levels of nesting.

Never have methods with more than five parameters. Try to have no more than three parameters at most being passed into a method, and generally only two.

Avoid having methods with out parameters.

Avoid having methods with ref parameters.

Avoid writing overloads for methods that you might use some day. Follow YAGNI principle.

Never programmatically click a button to execute the same action you have written in the button click event. Rather, call the same method which is called by the button click event handler.

Never hardcode a path or drive name in code. Get the application path programmatically and use relative path.

Never assume that your code will run from drive "C:". You may never know, some users may run it from network or from a "Z:".

Always in the application start up, do some kind of "self check" and ensure all required files and dependencies are available in the expected locations. Check for database connection in start up, if required. Give a friendly message to the user in case of any problems.

Always if a wrong value found in the configuration file, application should throw an error or give a message and also should tell the user what are the correct values.

Always return empty collections instead of null.

Always remove dead code.

Always remove unnecessary using statements.

Filenames and directory names are PascalCase, e.g. MyFile.cs.

General Guidelines (cont)

Where possible the file name should be the same as the name of the main class in the file, e.g. MyClass.cs.

In general, prefer one core class per file.

A maximum of one statement per line.

A maximum of one assignment per statement.

Indentation of 2 spaces, no tabs.

Column limit: 100.

No line break before opening brace.

No line break between closing brace and else.

Braces used even when optional.

Space after if/for/while etc., and after commas.

No space after an opening parenthesis or before a closing parenthesis.

No space between a unary operator and its operand. One space between the operator and each operand of all other operators.

Where possible, group interface implementations together.

Namespace using declarations go at the top, before any namespaces. using import order is alphabetical, apart from System imports which always go first.

Modifiers occur in the following order: public protected internal private new abstract virtual override sealed static readonly extern unsafe volatile async.

For single line read-only properties, prefer expression body properties (=>) when possible.

For everything else, use the older { get; set; } syntax.

Data Types

Never use magic numbers or magic strings. Use constants or enums.

Try to only use var when the type is very obvious.

Never compare floating point values using ==, !=, or Equals

Try to make all your variables and return values strongly typed.

Always use the decimal type for currency data. Floating point types can lead to inconsistent comparisons and rounding.

Object Oriented Programming

Always separate concerns. Separate User Interface Logic, Business Logic, Data Logic, into separate projects

Always create methods to do only 'one job'. Do not combine more than one job in a single method, even if those jobs are very small.

Always prefer composition over inheritance.

Avoid Premature Generalization. Create abstractions only when the intent is understood.

Avoid creating methods that function differently based on the underlying type of the object.

Always declare all data members private.

Always prevent instantiation of a class if it contains only static members.

Always explicitly define a protected constructor on an abstract base class.

Try to make all types internal by default.

Try to use using statements instead of fully qualified type names.

Never hide inherited members with the new keyword.

Always override the GetHashCode method whenever you override the Equals method.

Always override the Equals method whenever you implement the == operator, and make them do the same thing.

Always implement operator overloading for the equality (==), not equal (!=), less than (<), and greater than (>) operators when you implement IComparable

Always create variants of an overloaded method to be used for the same purpose and have similar behavior.

Always allow properties to be set in any order.

Never create a constructor that does not yield a fully initialized object

Comments

Try to not check in commented code. Trust source control.

Always put comments in English.