Coding Standards

This document describes the coding standards to be used for all code you write in this course.

1. Capitalization

There are three capitalization styles we'll use as we "name stuff" in our programs:

Name	Description	Example
Camel Case	The first word starts with a lower case letter and all other words start with a capital letter. The rest of the letters are lower case.	weaponDamage
Pascal Case	All words start with a capital letter, with the rest of the letters in lower case.	NonPlayableCharacter
Uppercase	All letters are capital letters. Words are separated by underscores(_).	SCREEN_WIDTH

The table below shows the proper capitalization style for the names of each kind of entity you'll have in your programs:

Entity	Capitalization
Class or Struct	Pascal Case
Field	Camel Case
Method	Pascal Case
Parameter	Camel Case
Property	Pascal Case
Local Variable	Camel Case
Constant	Uppercase
Enumeration	Pascal Case
Exception	Pascal Case, suffixed with Exception
Interface	Pascal Case, prefixed with I

We will NOT be using Hungarian notation (even though some books do), so use reasonable names for the entities in your programs and don't start those names with character(s) indicating the data type of that entity.

Note: the variables that you're declaring in your Main method (and all the other methods you write) are local variables. You should consistently follow this standard.

2. Commenting

Documentation comment blocks (documentation comments start with `///`) are used to describe the entities listed below. XML tags are used within the documentation comments.

Classes, Structures, and Interfaces

Comment every class, structure, or interface with a documentation comment block describing the purpose of the class or structure. The comment block must start with the line `///<summary>`and end with the line `///</summary>`.

Example

```
/// <summary>
/// A deck of cards
/// </summary>
public class Deck
```

Methods

Comment every method with a documentation comment block describing the purpose of the method, the parameters passed in to the method (if there are any), and the value returned by the method (if there is one).

The purpose of the method is described in a summary, which should always come first, so the comment block for a method must start with the line `///<summary>`. If the method doesn't have any parameters or return anything, it ends with the line `///</summary>`.

Parameters are described using the `param` tag. The `name` attribute should be identical to the parameter name. The text between `<param name="…">` and `</param>` describes what the parameter is used for. Each parameter gets their own `param` tag in the comment block.

The return value for the method is described using the `returns` tag. The text between `<returns>` and `</returns>` describes the return value.

Example 1

```
/// <summary>
/// The standard Main method
/// </summary>
/// <param name="args">command-line args</param>
static void Main(string[] args)
```

Example 2

```
/// <summary>
/// Draws the specified card from the deck
/// </summary>
/// <param name="location">the location of the card in the deck</param>
/// <returns>the card from the given location</returns>
public Card DrawCard(int location)
```

Properties

Comment every property with a documentation comment block describing the purpose of the property. The comment block must start with the line `///<summary>` and end with the line`///</summary>`.

Example

```
/// <summary>
/// Gets the number of cards in the deck
/// </summary>
public int NumCards
```

Line Comments

Every few lines of C# code you write should have a comment above them explaining what those lines are supposed to do. It's not necessary to provide a line comment for every line of code, because that actually just ends up making things more confusing. There are no set rules about how much or little you should comment your code with line comments, but providing a line comment every 3-5 lines of code is probably a good rule of thumb.

The format you should use for line comments is:

blank line or open curly brace above line comment
// line comment
code being commented (no blank line between comment and code

Example

```
// extract name and percent
string name = csvString.Substring(0, commaLocation);
float percent = float.Parse(csvString.Substring(commaLocation + 1));

// print name and percent
Console.WriteLine("Name: " + name);
Console.WriteLine("Percent: " + percent);
```

By doing it this way, you'll have easily-distinguished areas of line comments and the code described by the line comments.

3. Indentation

Indentation makes it easier to understand the structure of the code. Use the default indentation provided in Visual C# Express, which is 4 spaces inside each block of code.

Example (comments excluded for brevity)

```
public class ExampleClass
{
    static void Main()
    {
        Console.WriteLine("Coding standards Rock You Like a Hurricane!!");
    }
}
```

4. White Space

White space makes our programs much easier to read and understand. While it's difficult to enumerate all the examples of good use of white space, use the code I provide from class as guidance.

One example, however, is that a single-line comment should ALWAYS have a blank line preceding it.

5. Variable Names and Declarations

Unless you're using standard single-letter variables like i in a for loop or well-known acronyms like gpa, your variable names should consist of sequences of complete words, not just the first letter from each word. For example, goldPerMinute is a good variable name, while gpm isn't. Believe me, this is a huge help when you go back to code you wrote six months ago; even though you won't do that with the code you write here, you might as well just do it correctly from the start. Although C# lets you declare multiple variables on a single line, you should only declare one variable per line in this class.

(Only apply this rule AFTER Week 1 since some people have already completed Programming Assignment 1): Constants should be named with all capital letters and underscores between the words; for example, MINUTES_PER_HOUR.

6. String Variables

We can use either `string` (lower case) or `String` (upper case) to declare string variables in C#. You should use whichever you prefer when declaring string variables. Make sure your class includes `using System;` at the top.

7. Statement Length

You can write statements that are arbitrarily long in Visual C# Express (and most other IDEs), but that makes your code incredibly hard to read. Each statement you write must fit in the width of the default Visual C# Express editing window. If you have a statement that won't fit, break it up into multiple lines using concatenation (for strings) or at reasonable places in your arithmetic operations.

Example

```
Console.WriteLine("Four score and seven years ago, " +
    "People thought rock and roll was evil, " +
    "And digital computers didn't even exist!");
```

by Dr. Tim "Dr. T" Chamillard
University of Colorado