**ASSIST/UNA Coding Standards**

Version 1.0

March 10, 2014

Andrew Hamilton

*Software Quality Assurance*

University of North Alabama

CS 455

Software Engineering Project

Spring 2014

Contents

1.) General 2

2.) Naming Standards 2

3.) Headers 2

4.) Comments 3

5.) Class Structure 3

6.) Block Structure 3

7.) Expressions 4

8.) Example 5

**1.) General**

All testing guidelines will be uploaded to the team’s GitHub for developers to reference when needed. Developers will use the default text editing settings in Visual Studio 2013 for text font and size. All developers will ensure their font settings are set to Consolas size ten. Any given line of code will not exceed column 100. Indentations will be four spaces, the default in Visual Studio 2013. Only one statement will be written on each line. Naming of files will use the default C# method of using the class name for the file name. For example, the file containing the class ExampleClass will be named ExampleClass.cs. A file will contain only one class. An example file with correct coding standards is given in Section 8 and can also be found on the team’s Team Foundation Server. Each developer will read the coding standards and review the example before any code is written.

**2.) Naming** **Standards**

Class and method names will use the Pascal case naming convention. For example:

class ExampleClass

public void ExampleMethod() {}

The names to be used for classes and methods will be provided to each developer in the Detailed Design.

Variable / data member names will use the camel case naming convention. For example:

int exampleVariable;

private bool exampleVariableTwo;

Some of the variable and data member names will be provided in the Detailed Design. Developers will be allowed to use their judgment in naming variables and data members that are not provided in the Detailed Design. Any constant variables or data members will be named using all capital letters with words separated by underscores. For example:

int MAX\_SIZE = 100;

**3.) Headers**

Each class will contain a header which will include the following information: the name of the class, a description of the class, a table of modification history including the modification date, initials of the developer, and the changes that were made. An example class header is given in Section 8.

Each method will contain a header including the name of the method, the author(s), the input (parameters), what will be returned, and a description of the method’s function and purpose. If the method does not use input or have a return value, the field will contain “N/A.” The description field will contain an overview of the method, any non-returned output, such as changes to the screen, and any changes to pass-by-reference variables. An example method header is given in Section 8.

**4.) Comments**

Block comments will look like the following:

/\*

\* Line one.

\* Line two.

\* Last line.

\*/

Note that each line contains an asterisk; this is inserted by default in Visual Studio 2013. Each comment line will begin with a capital letter and each sentence will end with a period.

Single line comments will look like the following:

/\* This is a single line comment. \*/

Note that there is a space after the first asterisk and before the second asterisk. All comments will be written on the line above the code. No comments will be written on the same line as the code. Comments will not exceed column 100.

**5.) Class Structure**

The order of data members and methods for each class will be as follows:

1. constants
2. public data members
3. protected data members
4. private data members
5. public methods
6. protected methods
7. private methods

Data members and methods should be ordered alphabetically by their type first, then by their name. An example class is given Section 8. Developers will use the example as a template to ensure each class has the same structure and the same amount of whitespace.

**6.) Block Structure**

Curly braces for each block will be written with the following structure:

while(true)

{

/\* Statements go here. \*/

.

.

}

Note there is no whitespace after the opening brace or before the closing brace. Curly braces will not be used for blocks with one statement. Developers will decide where to use whitespace in each block. Any variable declarations in a block will be at the beginning of the block. Variables declared in each block will be ordered alphabetically first by the type name, then by the variable name.

If-else blocks will use the following form:

if(true)

{

/\* Statements go here. \*/

.

}

else

/\* Statement goes here. \*/

Note that there is a blank line in between the blocks. Also note there are no curly braces used in the else block with one statement, but braces will be used when there is more than one statement. More examples of blocks are given in Section 8.

**7.) Expressions**

Expressions using parentheses, including if statements, method parameters, and while statements, will use the same form as the following examples:

Print(int a, int b, int c);

if(x)

while(a == b)

Note that there are no spaces around the parentheses and method parameters are separated by spaces. Elements of expressions will be separated with spaces. For example:

a = b + c;

f = (a + b) \* c;

**8.) Example**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Name: ExampleClass

\*

\* =====================================================================================

\*

\* Description: This class is here to serve as an example for coding standards. The

\* programmers will adhere to the commenting standards that are evident in this

\* example

\* =====================================================================================

\*

\* Modification History

\* --------------------

\* 03/08/2014 THH Original version.

\* 03/09/2014 AAH Added awesomeness.

\* 03/13/2014 THH Reformatted some of the commenting blocks to match the max

\* length of the stars. The max length of commenting is up to column 100.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

namespace Processing

{

class ExampleClass

{

/\* Constants. \*/

public const string EXAMPLE\_PUBLIC\_CONSTANT = "This is a public constant.";

protected const string EXAMPLE\_PROTECTED\_CONSTANT = "This is a protected constant.";

private const string EXAMPLE\_PRIVATE\_CONSTANT = "This is a private constant.";

/\* Public members. \*/

public bool examplePublicBoolean;

public int examplePublicInteger;

/\* Protected members. \*/

protected int exampleProtectedInteger;

/\* Private members. \*/

private int examplePrivateInteger;

/\* Public methods. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Name: Main

\*

\* Author(s): First Last

\* First Last

\*

\* Input: N/A

\* Return: N/A

\* Description: The description of the method goes here.

\* It also continues here. (include any non-returned output,

\* i.e. printed to, pass-by-reference changes, etc.)

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public static void Main()

{

/\* Variable declarations. \*/

int input;

/\* Single line comments are formatted like this. \*/

Console.WriteLine("Hello World!");

/\*

\* Block comments are formatted like so.

\* All programmers should follow this format.

\* Last comment line.

\*/

Console.WriteLine("Press any key to continue.");

Console.ReadKey();

Console.Write("Enter an integer: ");

input = Convert.ToInt32(Console.ReadLine());

ExampleClass temp = new ExampleClass();

temp.ExamplePrivateMethod(4);

temp.Print(input);

Console.WriteLine("Press any key to continue.");

Console.ReadKey();

bool hello = temp.ExamplePrivateMethod(5);

Base16Practice();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Name: Base16Practice

\*

\* Author(s): First Last

\* First Last

\*

\* Input: N/A

\* Return: N/A

\*

\* Description: This method works just to practice with converting integers

\* to base 16 and back. The examples are printed out to the

\* console.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

private static void Base16Practice()

{

int val = Convert.ToInt32("00000026", 16);

Console.WriteLine("00 00 00 26 converted to integer: " + val);

string stringVal = val.ToString("X");

Console.WriteLine("Val converted back to hex (before): " + stringVal);

stringVal = stringVal.PadLeft(8, '0');

Console.WriteLine("Val converted back to hex (after): " + stringVal);

Console.ReadKey();

}

/\* Protected methods. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Name: Print

\*

\* Author(s): Travis Hunt

\*

\* Input: Number to be returned. (include slight

\* description of input)

\* Return: True/False

\*

\* Description: This method prints the number that was passed to it to the

\* console. (include any non-returned output, i.e. printed to

\* screen, pass-by-reference changes, etc.).

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

protected bool Print(int numberOne)

{

/\* Loop from 0 to numberOne, printing each increment \*/

for (int i = 0; i < numberOne; i++)

Console.WriteLine(i + " ");

Console.WriteLine();

/\* Brief description of what will trigger the if and what will occur \*/

if (numberOne < 10)

Console.WriteLine();

/\* Brief description of what will trigger the else and what will occur \*/

else

{

numberOne = 10;

Console.WriteLine(numberOne);

}

return true;

}

/\* Private methods. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Name: ExamplePrivateMethod

\*

\* Author(s): First Last

\* First Last

\*

\* Input: Number to be returned. (include slight description of input)

\* Return: True/False

\*

\* Description: This method prints the number that was passed to it to the

\* console.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

private bool ExamplePrivateMethod(int numberOne)

{

Console.WriteLine(numberOne);

return true;

}

}

}