CNS 1400 Style Guidelines

One of the goals of modern software engineering is to craft programs so that they are well designed, readable, and maintainable. Programming style guidelines go a long way toward insuring that this is the case. Most large programming shops have such guidelines and they will require you to follow them. In this course you will be expected to follow these guidelines. You may lose points when programs do not meet these gudelines.

Identifiers

Use names that have semantic meaning. Avoid single character, very short, or very long names.

Examples:

Semantic Names	Baffling Names Six Months From Now
amount	a
isFinished	x1

Follow the following C# language naming conventions.

Constants

All upper case with words separated by an underscore

Examples:

SIZE
CONVERSION_FACTOR

Class Names

Title case (capitalization of the first letter in each word)

Examples:

```
SimpleCalc
ActionListener
```

Variable names

Lower case for the first word and title case for every word thereafter.

Examples:

```
  name
  makeDeposit( )
```

Method Names

Title case (capitalization of the first letter in each word)

Examples:

```
  CalcPay( )
  MakeDeposit( )
```

Reserved words

Reserved words will be all lower case

Examples:

```
  true
  class
```

Block Delimiting and Indentation

Braces

Even though the C# language does not always require braces for some statements it is good programming practice to provide them. Use braces liberally to visually delimit the beginning and end of code blocks. Including braces now avoids the possibility of errors creeping into your code when you add additional statements at the last minute.

Place the opening (left) brace **{** so that it lines up with the left side of class header, function header, conditional statement, or repetitive statement. Place the closing (right) brace **}** in the same column as the opening brace. Always enter braces in opening/closing pairs to avoid forgetting to add one or the other or both. For braces that span more than three to five lines, comment the ending brace to indicate its nature (e.g., *// end if*).

Indentation

As you move from block to block, indent a minimum of two spaces. A good text editor will assist in the necessary tabbing.

Spacing

Although the C# compiler ignores most spacing, placing white space around code elements makes reading code easier.

Binary Operators

One space before and after the binary operator.

Examples:

```
2 + 2

945 >= 234
```

Parameter Lists

One space after the comma that separates each parameter.

Example:

```
drawString(drawingSurface, 25, 50);

Checkbox( "large", sizeGrp, true);
```

Parentheses

One space before or after the adjoining parenthesis when you have series of nested parenthesis, otherwise no space is necessary.

Comments

It is expected that your programs are adequately commented. A good approach to commenting your program is to start out by writing pseudo-code for your program, and putting that pseudo-code into your source code file(s) as comments. Then write the actual code that implements each line of pseudo-code.

In-Line Comments

Use in-line comments (//) for most of your commenting. Semantic identifiers go a long way toward eliminating the need for most comments. In-line comments above your code are easier to maintain. However, in-line comments on the same line are best for documenting a brace.

Multi-Line

Use the multi-line comment (/* */) to "comment out" code for debugging purposes.

File Prologues

All source code files should include the following file prologue. The file prologue clearly identifies the owner of the file and states what is in the file.

// Author: *studentName*
// Assignment: *projectNumber*
// Instructor: *InstructorName*
// Class: CNS *1400* Section: *sectionNumber*
// Date Written: *the Date*
// Description: *the description of what's in this file*

where the Name in ***studentName*** is your name, Number in ***projectNumber*** is the number of the assignment being submitted, Name in ***instructorName*** is your instructor's name, and ***sectionNumber*** is your three-digit section number. Description should include a brief overview of the contents of the file.

Example:

// Author: *Carl Coder*
// Assignment: *Project #1*
// Instructor: *Dr. deBry*
// Class: *CNS 1400 Section: 002*
// Date Written: *Sep 3, 2009*
// Description: *Non-interactive four-function integer*
// *calculator. Designed as a stand-alone*
// *C# application that displays results*
// *in a DOS console window*

Declaration

Every program that gets submitted must include the following text as part of the file prologue. Be sure that you read and understand what it says. Projects submitted without this declaration will be returned without being graded. A student may add this declaration and re-submit the assignment, but an appropriate late penalty will be added.

//I declare that the following source code was written solely by me.
//I understand that copying any source code, in whole or in part,
// constitutes cheating, and that I will receive a zero on this project
// if I am found in violation of this policy.

Class Header

Briefly describe the purpose of a class with commented lines just preceding the class definition. The following example is for a *Student* class.

```
// This class models a Student. It contains key information
// about a student and methods to manage that data
class Student
{
   . . .
}
```

Method Prologue

Use the following header to document the purpose of each method you write.

// Purpose: Brief sentence describing the purpose of the method.
// Parameters: List the method's parameters
// Returns: What does the method return
// Pre-conditions: Conditions that must exist for method to work
// Post-conditions:Conditions that exist after method has executed
// --

Example:

A sample method header for a writePhoneBook( ) method.

// Purpose: Writes the phonebook data to a file.
// Parameters: none
// Returns: none
// Pre-conditions: A valid filename has been provided
// Post-conditions: none
void writePhoneBook( );

Magic Numbers

A magic number is any numeric literal other than 1, 0, or -1 used in your program. However if 1, 0 and -1 are used to represent something other than the integers 1, 0, or -1 they will be considered magic numbers. Unfortunately, most code you will see in Java books or programming books in general will include magic numbers because it's easier to code in the short run. In the long rung, six months from today, you will be clueless as to what the number means. Therefore, DON'T USE MAGIC NUMBERS in your assignments.

So how do you avoid magic numbers? Define constants in you class or methods using semantic identifiers for each magic number.

Example:

```
  static const int SIZE = 10;
  TextField nameTxf = new TextField( SIZE );
```

Output

It is expected that your program's output will be neatly displayed and conform to whatever specifications are given in the assignment. All program output should be clearly labelled, and your spelling, punctuation, and grammar should be correct.

Every program should display your name, course and project/lab number when it starts.

If your program requires input from the user, the user should see a prompt that accurately reflects what is expected.