Naming and commenting

Cpt S 321

Washington State University

Namespaces

- They help us to organize large code
- The “global” namespace is the root namespace
  - E.g., “global::System” refers to the .NET System namespace
- The “using” directive:
  - Allows us to use the **types** in a namespace so that we do not have to qualify the access with the name.
    - E.g., “using System.Text;” allows us to say “StringBuilder sb;” as opposed to “System.Text.StringBuilder sb;”
  - Allows us to access **static members and nested types of a type** without having to qualify the access with the type name.
    - E.g., “using static System.Console;” allows us to say “WriteLine(“that’s quick!”);” as opposed to “System.Console.WriteLine(“too long!”);”
  - Allows us to create **an alias for a namespace or a type**.
    E.g., “using AliasToMyClass = NameSpace1.MyClass;” allows us to say “AliasToMyClass c;” as opposed to “NameSpace1.MyClass c;”

Namespaces (cont.)

- The “using” directive has a scope
  - E.g., if defined in the beginning of a file, then it is limited to the file in which it appears
  - E.g., if defined in a namespace, then it is limited to that namespace

- You can define the same namespace in more than one place, e.g.,

  namespace MyCompany.Proj1 { class MyClassA { … } }

  namespace MyCompany.Proj1 { class MyClassB { … } }

- We can have nested namespaces using the “.” operator, e.g., “namespace1.namespace2”
  - Ex., in our HelloWorld example, we can have namespaces “HelloWorld.Math” in which we define the Angle class and “HelloWorld.DataStructures” in which we define the LinkedList.

Namespaces – Example 1

- What are the fully qualified names of the entities?

```
namespace N3
{
    class C3
    {
        public static void Main(string[] args)
        {
            // TODO: Create an instance of C2 (defined in N1)


            // TODO: Create an instance of C2 (defined in N2)


        }
    }
}
```

```
namespace N1
{
        class C1
        {
                internal class C2
                {

                 }
        }

        namespace N2
        {
                class C2
                {

                 }
        }
}
```

Namespaces – Example 1 solution

- What are the fully qualified names of the entities?

```
namespace N3
{
    class C3
    {
        public static void Main(string[] args)
        {
            // Create an instance of C2 (defined in N1)
            N1.C1.C2 aC2Instance;

            // Create an instance of C2 (defined in N2)
            N1.N2.C2 anotherC2Instance;

        }
    }
}
```

```
namespace N1
{
        class C1
        {
                internal class C2
                {

                 }
        }

        namespace N2
        {
                class C2
                {

                 }
        }
}
```

Namespaces – Example 2

```
namespace N1
{
    public class C1
    {
        public string Name { get; set; }

        public static void SayHello()
        {
            Console.WriteLine("Hello");
        }

        public void SayBye()
        {
            Console.WriteLine("Bye ” + this.Name);
        }
    }
}
```

```
namespace N2
{
    public class C2
    {
        static void Main()
        {
            // how do we call SayHello() here?


            // how do we call SayBye() here?



        }
    }
```

Namespaces – Example 2 solution

```
namespace N1
{
    public class C1
    {
        public string Name { get; set; }

        public static void SayHello()
        {
            Console.WriteLine("Hello");
        }

        public void SayBye()
        {
            Console.WriteLine("Bye ” + this.Name);
        }
    }
}
```

```
namespace N2
{
    public class C2
    {
        static void Main()
        {
            // how do we call SayHello() here?

            N1.C1.SayHello();

            // how do we call SayBye() here?

            N1.C1 anInstanceC1= new C1();
            anInstanceC1.SayBye();

        }
    }
```

Naming – common sense

- Concise and consistent
- Must reflect the functionality
- **Important**: Naming is part of the grading criteria for HWs!

Lexicon Bad Smells (LBS): poor naming practices that we MUST avoid

- Extreme contraction
  - ✗ aSz
  - ✓ arraySize
  - Exceptions: commonly used abbreviations such as http, sql, …

- Inconsistent (or ambiguous) identifier use

```
public class Document
{
    private string absolutePath;
    private string relativePath;
    private string path;   // ✗ inconsistent identifier
}
```

Lexicon Bad Smells (LBS) (cont.)

- Meaningless terms

  ✗ foo, bar, a, b, c, i, j, myMethod, myClass, etc.
  Exceptions: i and j are acceptable as indexes in short/simple loops

- Misspelled identifiers

- Odd grammatical structure:

```
public abstract class Compute // ✗ compute is a verb
{
    public  abstract void Addition(); // ✗ addition is a noun
}
```

Lexicon Bad Smells (LBS) (cont.)

- Overloaded identifiers
  - ✗ CreateExportList
  - ✓ CreateList, ExportList
- Useless type indication
  - ✗ NameString
- Whole-part ambiguous identifiers

```
public class Account
{
    private string account; // ✗
}
```

Lexicon Bad Smells (LBS) (cont.)

- Synonyms/similar identifiers
  - ✗ Copy, Replica
- Wrong context
  - ✗ Having a namespace Detectors and a namespace Collections and declare TypeDetector in Collections
- No hyponym/hypernym in class hierarchies
  - ✗ Declare class UndergraduateStudent as a subclass of Item
- Not following standard naming conventions adopted for the project/company

Other lexicon smells: Linguistic Antipatterns (LAs)

- The type of an entity is inconsistent with its name:
  - ✗ **void get**MethodBodies(…){…}
  - ✗ public **void isValid**(…){…}
  - ✗ public **void check**Collision(…){…}
  - ✗ **void get**MethodBodies (…){…}
  - ✗ public **Dimension set**Breadth
  - ✗ public static Control**Enable**State **disable**(Control w) {…}
  - ✗ public **boolean** get**Stats**() {
  - ✗ MAssociationEnd start;
  - ✗ Vector _target;
  - ✗ boolean _stats

Linguistic Antipatterns (LAs) (cont.)

- The comment of an entity is inconsistent with its declaration/implementation

  ✗ Poor practice:

```
/**
* Configuration default exclude pattern,
* ie .*\/@href|.*\/@action|frame/@src
*/
public static String INCLUDE_NAME_DEFAULT
        = ".*/@href=|.*/@action=|frame/@src=";
```

Linguistic Antipatterns (LAs) (cont.)

- The comment of an entity is inconsistent with its declaration/implementation (cont.)

  ✗ Poor practice:

```
/**
* Returns true if this listener has a target for a
* back navigation. Only one listener needs to return
* true for the back button to be enabled.
*/
public boolean isNavigateForwardEnabled() {…}
```

Other examples of naming conventions

- Use **camelCasing** for method arguments and local variables
  - ✓ logEvent (parameter name)
  - ✓ itemCount (local variable declared in a method)
- Use **PascalCasing** for class names and method names
  - ✓ Angle (class/constructor name)
  - ✓ AddItem (method name)
- Avoid _
  - ✗ item_count
  - ✓ itemCount

Other examples of naming conventions (cont.)

- Use the “I” prefix for interfaces
  - ✓ IShape
- Use namespaces to organize your code
  - ✓ HelloWorld.DataStructures
  - ✓ HelloWorld.IO
- Check MSDN, for example, check the:
  - General naming conventions
  - Capitalization conventions
- Use the Microsoft’s .NET Framework as an example

Commenting

- Automatic templates for entities are generated for you by typing “///” above the entity

- Example

```
/// <summary> /// The main Math class.
/// Contains all methods for performing basic math functions.
/// </summary>
public class Math {
    /// <summary>
    /// Adds two integers and returns the result.
    /// </summary>
    public static int Add(int a, int b) {
        // If any parameter is equal to the max value of an integer
        // and the other is greater than zero
        if ((a == int.MaxValue && b > 0) || (b == int.MaxValue && a > 0))
            throw new System.OverflowException();
        return a + b;
    }
}
```

Leading (summary) comment:
Explains the “what”

Inline comment:
Explains the “how”

Commenting

- More examples:

https://docs.microsoft.com/en-us/dotnet/csharp/codedoc - put-it-all-together

- **Important:**
  - Commenting is part of the grading criteria for HWs
  - Which entities should be commented? Every **type**, **method**, **attribute**, and **test case** must be commented.
  - What information is required in the comments? Entities MUST have at a minimum: **<summary>**, **<returns>**, **<param>**, and **<exception>**, when applicable.
  - Commenting MUST be done while coding, not at the end

How do we enforce consistency?

- For this course, we will use **StyleCop**
  - Check the Current analyzers

- Installation
  - Check the installation guide on the Canvas website. Have it installed before our next class – come to office hours before class in case of problems!

- **Important**:
  - You MUST use a StyleCop configuration for all your HWs.
  - You will start with the **default configuration** and only modify it to 1) disable contradicting rules or 2) adapt it to your coding preferences (must be justified and documented).
  - Every change you make to the configuration MUST be justified in a README file in your repository.

Summary: for all homework assignments

- You will follow naming conventions as discussed here. In addition, feel free to define your own style. Key point: be consistent!

- You will document your code with
  - leading comments (for all classes, methods, fields, properties)
  - inline comments when appropriate (when the code is complex)

- Your repositories must contain the StyleCop files and a justification of the changes in the README file of your repo