CSC322

Lecture Note 4

Dr. Victor Odumuyiwa

Learning Outcomes

At the end of this class, you should be able to:

- Write specifications for your programs
- Auto-generate documentation

SPECIFICATIONS

Specification

- Many of the nastiest bugs in programs arise because of misunderstandings about behaviour at the interface between two pieces of code.
- Every programmer has specification in mind but not all programmers write them down.
- As a result, different programmers in a team have different specifications in mind

Specifications contd.

- A specification of a method can talk about the parameters and return value of the method, but it should never talk about the local variables of the method or private fields of the method's class.
- You should consider the implementation invisible to the reader of the specification.
- A specification answers "what questions" and not "how questions"

Specification structure

- A specification of a method consists of several clauses:
  - A precondition indicated by the keyword ***requires***
  - A postcondition, indicated by the keyword ***effect*** or ***ensure***

Specification structure contd.

- An assertion is a logical expression, which, if false, indicates an error. A logical expression is an expression of type boolean.
- A precondition of an operation is an assertion which must be true just before the operation is called.
- A postcondition of an operation is an assertion which must be true just after the operation has been completed.

Specifications contd.

- Deterministic vs. Underdetermined
  - does the spec define only a single possible output for a given input, or allow the implementer to choose from a set of legal outputs?
- Declarative vs. Operational
  - does the spec just characterize what the output should be, or does it explicitly says how to compute it?
- Strong vs. Weak
  - does the spec have a small set of legal implementations, or a large set?

Specifications contd.

```
public static int FindA(int[] a, int  val)
        {
            for (int i = 0; i < a.Length; i++)
            {
                if (a[i] == val)
                {
                    return i;
                }
            } return a.length;
      }
```

```
public static int FindB(int[] a , int val)
        {
         for (int i = a.Length -1; i>=0;  i--)
            {
                if (a[i] == val)
                {
                    return i;
                }
            } return -1;
      }
```

Specifications contd.

static int find (int[]a, in val)
requires: val occurs exactly once in a
effects: returns index i such that a[i] = val

static int find (int[]a, in val)
requires: val occurs in a
effects: returns index i such that a[i] = val

static int find (int[]a, in val)
requires: val occurs in a
effects: returns largest index i such that a[i] = val, or -1 if no such i

Deterministic

Stronger vs. Weaker Specifications

A specification **X** is stronger or equal to a specification **Y** if:

- **X**'s precondition is weaker than or equal to **Y**'s
- **X**'s postcondition is stronger than or equal to **Y**'s, for the states that satisfy **Y**'s precondition

Implication

- You can always weaken the precondition (placing fewer demands on a client will never upset him)
- You can always strengthen the postcondition (it means making more promises)

Stronger vs. Weaker Specifications

```
static int Find1(int[]a, int val)
requires:  val occurs exactly once in a
effects: returns index i such that a[i] = val
```

```
static int FindStronger2(int[]a, int val)
requires:  val occurs at least once in a
effects: returns index i such that a[i] = val
```

```
static int FindStronger3(int[]a, int val)
requires:  val occurs at least once in a
effects: returns lowest index i such that a[i] = val
```

```
static int FindNew(int[]a, int val)
requires:  nothing
effects: returns index i such that a[i] = val or -1
if no such i
```

Designing Good Specifications

- succinct, clear and well-structured
- coherent (should have one purpose)
- strong enough
- weak enough
- don't use specification comments to explain implementation for a maintainer rather use comments within the body of the method

Writing Specifications in C#

- Use of XML Commenting
  - Helps to auto-generate program documentation
  - Provides intellisense within the code window
- Configuration
  - Right Click on the project in the solution explorer and select "Properties".
  - Within the properties dialog double click on the "Configuration Properties" node.
  - Click on the "Build" tab and check the "XML documentation file"
  - Give your documentation file a name
- Documentation
  - SandCastle (free and open source tool)

Tags for XML Commenting

Tags	Description
<c>	gives you a way to indicate that text within a description should be marked as code
<code>	gives you a way to indicate multiple lines as code. Use <c> to indicate that text within a description should be marked as code.
	The cref attribute in an XML documentation tag means "code reference." It specifies that the inner text of the tag is a code element, such as a type, method, or property.
<example>	The <example> tag lets you specify an example of how to use a method or other library member. This commonly involves using the <code> tag.
<exception>	The <exception> tag lets you specify which exceptions can be thrown.
<include>	The <include> tag lets you refer to comments in another file that describe the types and members in your source code.
<list>	define the heading row of either a table or definition list. <term><item><description>
<para>	use inside a tag, such as <summary>, <remarks> or <returns> and lets you add structure to the text.
https://msdn.microsoft.com/en-us/library/x640hcd2.aspx	

Tags for XML Commenting Contd.

Tag	Description
<param>	used in the comment for a method declaration to describe one of the parameters for the method.
<paramref>	gives you a way to indicate that a word in the code comments, for example in a <summary> or <remarks> block refers to a parameter.
<permission>	lets you document the access of a member.
<remarks>	used to add information about a type, supplementing the information specified with <summary>.
<returns>	used in the comment for a method declaration to describe the return value.
<see>	lets you specify a link from within text.
<seealso>	lets you specify the text that you might want to appear in a See Also section.
<summary>	used to describe a type or a type member.
<value>	lets you describe the value that a property represents.

Installing SandCastle

- Download and Install Sand Castle via the MSI.
- Download and Install the Sand Castle Help File Builder (SHFB) MSI by Eric Woodruff
- Download and Install / Patch the Sand Castle Styles
- Run SHFB
- Add documentation sources (csproj or dll)
- Add references (csproj or dll)
- Set the help file format and other SHFB options (I recommend using the MemberName naming method to get html links with names instead of Guids)
- Run SHFB build, wait and done!

http://blogs.msdn.com/b/msgulfcommunity/archive/2014/04/22/creating-documentation-in-c-using-visual-studio-and-sandcastle.aspx

Resources

- MIT 6.005 fall 2013 lecture notes
- http://www.codeproject.com/Articles/3009/C-Documenting-and-Commenting