CS 683 Emerging Technologies
Spring Semester, 2003
Doc 20 C# Intro
Contents

References

C# Language Specification, http://download.microsoft.com/download/0/a/c/0acb3585-3f3f-4169-ad61-efc9f0176788/CSharp.zip

Programming C#, Jesse Liberty, O'Reilly, Chapters 1-3

Compilers

Microsoft Visual Studio .NET

Mono http://go-mono.org/
Open source compiler and runtime for Linux

Shared Source Common Language Infrastructure (sscli)
Linux
Windows
Mac OS 10
Download
http://www.microsoft.com/downloads/details.aspx?FamilyId=3A1C93FA-7462-47D0-8E56-8DD34C6292F0&displaylang=en

Information
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/Dndotnet/html/mssharsourcecli.asp

C#

Another C-based object-oriented programming language

```
public class Hello
	{
	public static void Main()
		{
		//sample comment
		System.Console.WriteLine( "hi");
		/* another
		comment */
		}
	}
```

```
using System;
public class Hello
	{
	public static void Main()
		{
		Console.WriteLine( "hi");
		}
	}
```

Naming Convention

Pascal Casing

Capitalize the first character of each word

SomeClassName

Camel Casing

Capitalize the first character of each word except the first word

someVariableName

Convention

Type	Case	Notes
Class	PascalCase	
Attribute Class	PascalCase	Has suffix of Attribute
Exception Class	PascalCase	Has suffix of Exception
Constant	PascalCase	
Enum type	PascalCase	
Enum values	PascalCase	
Event	PascalCase	
Interface	PascalCase	Has a prefix of I
Method	PascalCase	
Namespace	PascalCase	
Property	PascalCase	
Public Instance Field	PascalCase	Rarely used
Protected Instance Field	camelCase	Rarely used
Parameter	camelCase	
Local variable	camelCase	

Word choice

Do not use abbreviations in identifiers

If you must use abbreviations, do use camelCase for any abbreviation containing more than two characters, even if this is not the usual abbreviation.

Classes

Do name classes with nouns or noun phrases

Do use PascalCase

Do use sparingly, abbreviations in class names

Do not use any prefix (such as “C”, for example)

Note no Hungarian notation

Do not use any underscores

Parameters

Do use descriptive names such that a parameter's name and type clearly imply its meaning

Do name parameters using camelCase

Do prefer names based on a parameter's meaning, to names based on the parameter's type

Do not use Hungarian-type prefixes

Methods

Do name methods with verbs or verb phrases

Do name methods with PascalCase

RemoveAll(), GetCharArray(), Invoke()

Case sensitivity

Don't use names that require case sensitivity

C# is case sensitive

Components might need to be usable from both case-sensitive and case-insensitive languages

Avoiding type name confusion

Different languages use different names to identify the fundamental managed types

In a multi-language environment avoid language-specific terminology

Do use semantically interesting names rather than type names

In the rare case that a parameter has no semantic meaning beyond its type, use a generic name.

For example use:

```
void Write(double value);
void Write(float value);
void Write(long value);
void Write(int value);
void Write(short value);
```

Rather than

```
void Write(double doubleValue);
void Write(float floatValue);
void Write(long longValue);
void Write(int intValue);
void Write(short shortValue);
```

Types

Built-in or base types
User-defined types

Value Types

Placed on stack

Base types, enum & struct

Reference Types

Placed on heap

Objects, arrays, delegate types

Built-in Types

Type	Description	Example
string	Unicode chars	string s = "hello";
sbyte	8-bit signed	sbyte val = 12;
short	16-bit signed	short val = 12;
int	32-bit signed	int val = 12;
long	64-bit signed	long val1 = 12; long val2 = 34L;
byte	8-bit unsigned	byte val1 = 12;
ushort	16-bit unsigned	ushort val1 = 12;
uint	32-bit unsigned	uint val1 = 12; uint val2 = 34U;

ulong	64-bit unsigned	ulong val1 = 12;
		ulong val2 = 34U;
		ulong val3 = 56L;
		ulong val4 = 78UL;

float	Single-precision	float val = 1.23F;
double	Double-precision	double val1 = 1.23; double val2 = 4.56D;
bool	Boolean type;	bool val1 = true; bool val2 = false;
char	Unicode char	char val = 'h';
decimal	Precise decimal type with 28 significant digits	decimal val = 1.23M;

Escape Sequences

Escape sequence	Character name	Unicode encoding
\'	Single quote	0x0027
\"	Double quote	0x0022
\\	Backslash	0x005C
\0	Null	0x0000
\a	Alert	0x0007
\b	Backspace	0x0008
\f	Form feed	0x000C
\n	New line	0x000A
\r	Carriage return	0x000D
\t	Horizontal tab	0x0009
\v	Vertical tab	0x000B

Unicode Escape Sequence

char sample = ‘\u0066’;

Keywords

abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	do	double	else	enum
event	explicit	extern	false	finally
fixed	float	for	foreach	goto
if	implicit	in	int	interface
internal	is	lock	long	namespace
new	null	object	operator	out
override	params	private	protected	public
readonly	ref	return	sbyte	sealed
short	sizeof	stackalloc	static	string
struct	switch	this	throw	true
try	typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual	void
volatile	while			

Keywords as Identifiers

int @int = 3;

Prefixing an identifier with @ allows keywords to be used

@ is not part of the identifer

@cat is the same identifier as cat

Implicit Conversion

Implicit cast to larger types

Must use explicit cast to smaller types

```
public static void Main()
    {
    char a = 'c';
    int b = a;
    a =(char) b;
    a = b; //Compile error
    int c = b;
    }
```

Definite Assignment

Must assign a variable before using it

Constants

Literals
Symbolic Constants
Enumerations

```
public class ConstantExample
	{
	public static void Main()
		{
		const int b = 5;
		b = 4; //Compile error
		}
	}
```

Enumerations

```
enum Value
        {
        First = 1,
        Second = 2,
        Third,  //gets value 2, indexing starts at 0
        Fourth = 4,
        }

class Hello
        {

        static void Main()
                {
                Value b = Value.Second;
                b = (Value) 1;
                b = b + 1;
                b++;
                System.Console.WriteLine( b);
                }

        enum Another :short
                {
                Foo = 2,
                Bar = 3,
                }
        }
```

Basic Control Structures

if

```
if (x == 5)
   y = 9;

if (z > 3)
   {
   y = 10;
   w = 1;
   }
else
    x = 2;
```

switch

```
const int Libertarian = 4;

int myChoice = Libertarian;
switch (myChoice)
   {
   case Democrat:
      Console.WriteLine("It all Bush's fault");
      break;
   case NewLeft:
      goto case Progressive;
   case LiberalRepublican:
   case Republican:
      Console.WriteLine("We rule");
      break;
   case Progressive:
      Console.WriteLine("Progress is best");
      break;
   default:
      Console.WriteLine("Odd ones out");
      break;   //Required
   }
```

Swtich on Strings

```
string myChoice ="Libertarian";
switch (myChoice)
        {
        case "Democrat":
                Console.WriteLine("It all Bush's fault");
                break;
        case "NewLeft":
                goto case "Progressive";
        case "LiberalRepublican":
        case "Republican":
                Console.WriteLine("We rule");
                break;
        case "Progressive":
                Console.WriteLine("Progress is best");
                break;
        default:
                Console.WriteLine("Odd ones out");
                break;
        }
```

goto

```
using System;

class GotoExample
	{

	static int Main()
		{
		int k = 0;
		aLabel:
		Console.WriteLine("K: " , k);
		k++;
		if (k < 10)
			goto aLabel;
		return 0;
		}
	}
```

While

```
using System;

class Hello
	{

	static int Main()
		{
		int k = 0;
		while (k < 10)
			{
			Console.WriteLine("K: " , k);
			k++;
			}
		return 0;
		}

	}
```

Do-While

```
using System;

class Hello
	{

	static int Main()
		{
		int k = 0;
		do
			{
			Console.WriteLine("K: " , k);
			k++;
			}
		while
			(k < 10);
		return 0;
		}

	}
```

for

```
using System;

class Hello
	{

	static int Main()
		{
		for(int k = 0; k < 10; k++)
			{
			Console.WriteLine("K: " , k);
			}
		return 0;
		}

	}
```

Continue & Break

break – exit the loop

continue – return to top of loop & start next iteration

```
using System;

class Hello
	{

	static int Main()
		{
		string input = Console.ReadLine();
		for(int k = 0; k < 10; k++)
			{
			if (0 == k % 2 )
				continue;
			if (5 == k)
				break;
			Console.WriteLine("K: " , k);
			}
		return 0;
		}
	}
```

Operators & Precedence

Rows are ordered by precedence

Operators in top row have highest precedence

Category	Operators
Primary	x.y f(x) a[x] x++ x-- new typeof checked unchecked
Unary	+ - ! ~ ++x --x (T)x
Multiplicative	* / %
Additive	+ -
Shift	<< >>
Relational and type-testing	< > <= >= is as
Equality	== !=
Logical AND	&
Logical XOR	^
Logical OR	|
Conditional AND	&&
Conditional OR	||
Conditional	?:
Assignment	= *= /= %= += -= <<= >>= &= ^= |=

Ternary Operator

```
class TernaryExample
	{

	static void Main()
		{
		int a = 5;
		int b = 7;
		int max = a > b ? a : b;
		}
	}
```

Preprocessor Directives

#define
#if
#elif
#else
#endif
#region
#endregion

Example

```
#define DEBUG

class PreprocessorExample
        {

        static void Main()
                {
                int a = 5;

                #if DEBUG
                int b = 7;
                #else
                int b = 1;
                #endif
                int max = a > b ? a : b;
                }

        }
```

Region

Used by editors to fold text to show only the comment

```
class RegionExample
	{

	#region
	/// <sumary>
	///Comment for the
	/// menthod below
	/// </summary>
	private void ComputeArea()
		{
		//blah
		}
	#endregion
	}
```

Namespaces

```
namespace CS683
	{
	public class Hello
		{

		public static void max()
			{
			int a = 5;
			int b = 7;
			int max = a > b ? a : b;
			}
		}
	}

class Test
	{
	static void Main()
		{
		CS683.Hello.max();
		}
	}
```

Nesting Namespaces

```
namespace CS683
        {
        namespace Examples
                {
                public class Hello
                        {
                        public static void max()
                                {
                                int a = 5;
                                int b = 7;
                                int max = a > b ? a : b;
                                }
                        }
                }
        }

class Test
        {
        static void Main()
                {
                CS683.Examples.Hello.max();
                }
        }
```