C# Notes

Blake McBride (blake@mcbride.name)

Mechanics

Files

.cs $\Longrightarrow$ .exe

Mono

```
Compiling:  mcs   <file>.cs
Compiling:  mcs   <file>.cs   /r:System.Windows.Forms.dll
Running:    mono   <file>.exe
Running:    <file>                  if .NET or Mono is installed

<file> does not have to be the same as the name of the
       class being defined internally.
```

Libraries are kept in DLL's. To create a DLL use:

```
mcs  -target:library  <file>.cs
```

Compile and use a DLL as follows:

```
mcs  -reference:file  <file>.cs
```

The same goes for modules using "–target:module" and "–addmodule:file"

Microsoft's C#

```
Compiling:  csc   <file>.cs
```

Hello World Program

```
class Test1 {
    static int Main(string[] args) {
        System.Console.WriteLine("Hello World!");
        return 0;
    }
}
```

Data Types

Type	Size in bytes	Example
bool	1	true/false
byte	1	0–255
sbyte	1	-128 – 127
char	2 (unicode)	'H'
short	2	
ushort	2	unsigned
int	4	
uint	4	unsigned
long	8	
ulong	8	
decimal	12	355M
float	4	
double	8	
string		"Hello"

null represents a null object reference.

Strings

Strings are immutable.

```
string x = @"a\b";   //  @ means literal - no escape sequences

str.Length //  The length of the string

str[x]  //  0 based indexing

str1 == str2  //  returns true/false depending on the actual
                  string bytes (not an object pointer compare)
str1 != str2

str1 + str2  //  concatenates the two strings
                 creating a new string
                 only one has to be a string, the
                 other will be converted .ToString
str1 += str2

int r = string.Compare(str1, str2)

r = string.Compare(str1, str2, true)  // case-insensitive

string.Compare(s1, i1, s2, i2, n, ci)  //  compare sub-strings with
                                           optional case ignore

str.Substring(startPos)
str.Substring(startPos, num)
str.ToLower()
str.ToUpper()

char[] carray = str.ToCharArray();
str = new string(carray);
```

Use *StringBuilder* class for mutable strings.

```
StringBuilder sb = new StringBuilder();

sb.Append("string");
```

Arrays

```
int[]  x;
x = new int[100];
x[0] = 44;
x.Rank
x.Length
int[]  y = new int[5] { 4, 3, 6 };
int[]  y = { 4, 3, 6 };
int[]  z = y;   //  z and y point to the same array

// rectangular multi-dimensional
double[,]  balance;
balance = new[33,44];
balance[4,16] = 4.55;

// Jagged multi-dimensional
int [][]  m = new int [2][];
m[0] = new int[7];
m[1] = new int[22];
m[1][12] = 6;
```

Classes

```
/*  Sample class definition and use  */

class Test2 {
    static int Main(string[] args) {
        MyClass i = new MyClass(4);
        System.Console.WriteLine(i.getIV());
        MyClass.setCV(88);
        System.Console.WriteLine(MyClass.getCV());
        return 0;
    }
}

class MyClass {
    private static int cv;   // class variable
    private int iv;          // instance variable

    public MyClass(int v) {  // constructor
        iv = v;
    }
    public static void setCV(int v) {
        cv = v;
    }
    public static int getCV() {
        return cv;
    }

    public int getIV() {
        return iv;
    }
}
```

Use "*this*" to the object being refered to.

Use "*base.method(args)*" to do super calls.

Use "*: base(args)*" in constructor headers to call the super constructor.

"*object*" is the base class.

Modifiers

```
Class:      [public/internal]  [abstract/sealed]  [partial]  class  <className>
            [:  [<superclass>]
            [, <interface>]...]  {  ...  }

            sealed = class cannot be subclassed
            internal = default


Inner Class:    [public/private/protected/internal]
            [abstract/sealed]  class  <className>
            [:  [<superclass>]  [, <interface>]...]  {  ...  }


Interface:[public/internal]   interface  <interfaceName>
            [:  <superinterface> [, <superinterface>]]... {  ...  }


Method:     [public/private/protected/internal]  [static/virtual/override]
            [abstract/sealed]  returnType  <methodName>
            (  [args] )  {  ...  }

            sealed = method cannot be overridden and no dynamic lookup


Field:      [public/private/protected/internal]  [static]
            [readonly/volatile/const]  <type>  <fieldName>;


Variable: [const]  <type>  <variableName>;

            sealed = value cannot change
```

"static" always means item is associated with the class instead of an instance. Default access for all class members is "private".

Namespaces

To put classes in a file into a namespace use the following.

```
namespace  <namespace>  {
    <class definitions>
}
```

To access a namespace use:

```
<namespace>.class  var;
```

Or use:

```
using <namespace>;

class  var;
```

Delegates (function pointers)

```
using System;
delegate int AddDelegate(int a, int b);  //  like a typedef

class Test {
    static int Main(string[] args) {
        Console.WriteLine(apply(add));
        return 0;
    }

    static int add(int a, int b) {
        return a + b;
    }

    static int apply(AddDelegate fun) {
        return fun(3, 4);
    }
}
```

Unusual Statements

"*foreach*" iterates over a collection. For example:

```
for (int i=0 ; i < ary.length ; i++)
    Console.WriteLine(ary[i]);
```

Is the same as:

```
foreach (int v in ary)
    Console.WriteLine(v);
```

Exceptions

```
try {
  // code to execute, can throw exception
}
catch (Exception-type ex) {
  // handle exception of exception-type, ex is exception object
}
catch (Exception-type) {
  // handle exception - no exception variable
}
catch {
  // catches all exceptions
}
finally {
  //  code to execute (after everything) whether exception or not
}
```

Exceptions must be of type *System.Exception*.

ex.StackTrace and *ex.Message* provide exception info as strings.

Use the following to throw an exception:

```
throw new ExceptionClass(args);
```

where *ExceptionClass* is one of the classes related to *System.Exception*.

LinkedList

```
LinkedList<string>  ll = new LinkedList<string>();

ll.AddFirst(str)
ll.AddLast(str)
ll.Count
ll.First.Value
ll.First.Next
ll.Last

foreach (string elm in ll)
       .....

ll.RemoveFirst()
ll.RemoveLast()
```