MVA jumpstart

03 | Features of C#, Part 2

Jerry Nixon | Microsoft Developer Evangelist
Daren May | President & Co-founder, Crank211

Microsoft

Module Overview

- Controlling Programmatic Flow
- Manipulating Types and Strings

Controlling Programmatic Flow

MVA Microsoft Virtual Academy

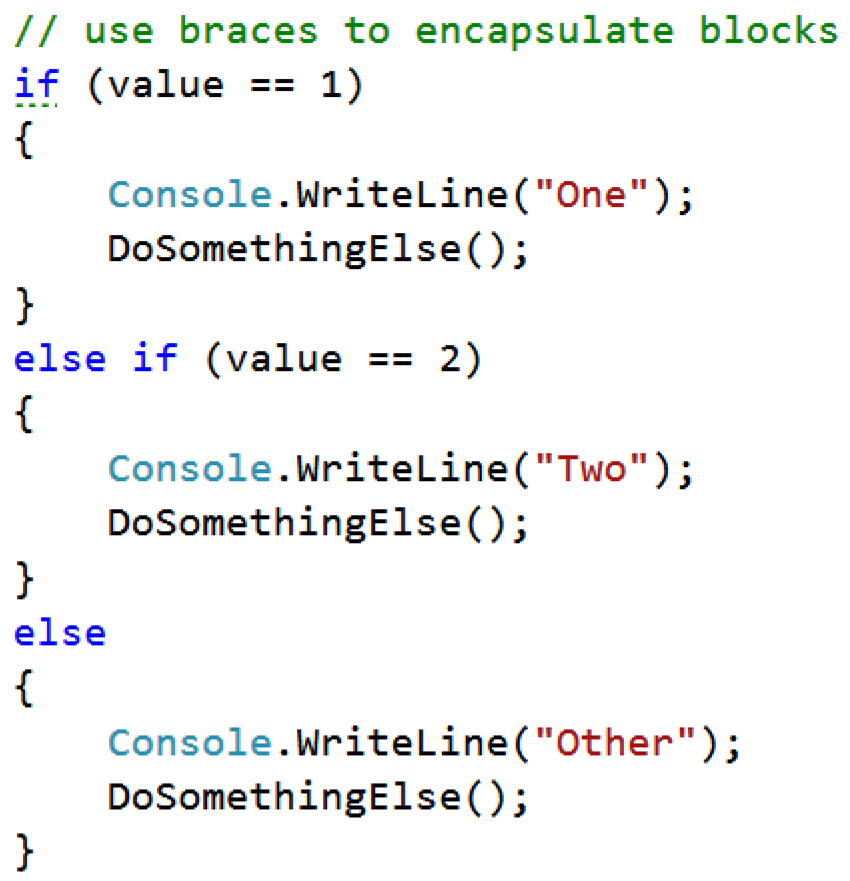
Microsoft

Controlling Flow

- Many statements impact program flow:
  - Selection statements
    - if, else, switch
  - Iteration statements
    - do, for, foreach, in, while
  - Jump statements
    - break, continue, default, goto, return, yield

Selection Statements : if

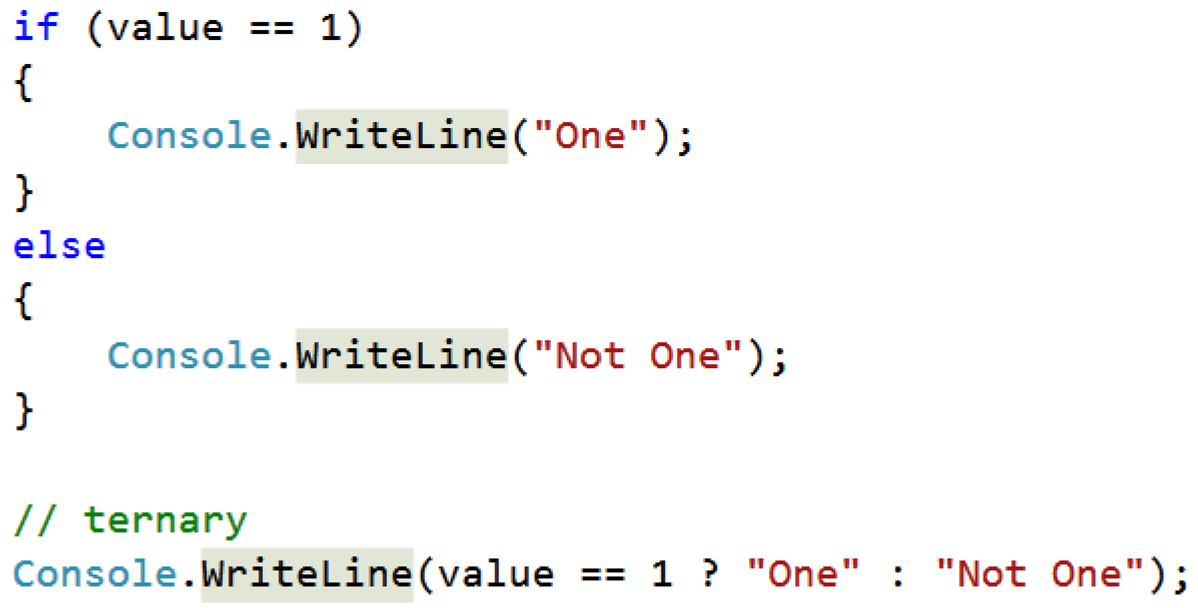
- Selection statements evaluate Boolean expressions and direct execution
- *If statements* can be nested within other *if statements*.



```
// use braces to encapsulate blocks
if (value == 1)
{
    Console.WriteLine("One");
    DoSomethingElse();
}
else if (value == 2)
{
    Console.WriteLine("Two");
    DoSomethingElse();
}
else
{
    Console.WriteLine("Other");
    DoSomethingElse();
}
```

Selection Statements :ternary

- The ternary or conditional operator can be used as *if statement* shorthand.



```
if (value == 1)
{
    Console.WriteLine("One");
}
else
{
    Console.WriteLine("Not One");
}

// ternary
Console.WriteLine(value == 1 ? "One" : "Not One");
```

DEMO

Microsoft

Switch (008)

Iteration Statements: while, do-while

- while and do-while statements execute a body of code if the expression evaluates to true.
  - *while* evaluates the expression before executing the body, so the body may execute 0 or more times.
  - *do-while* evaluates the expression after the first execution of the body, so the body executes at least once

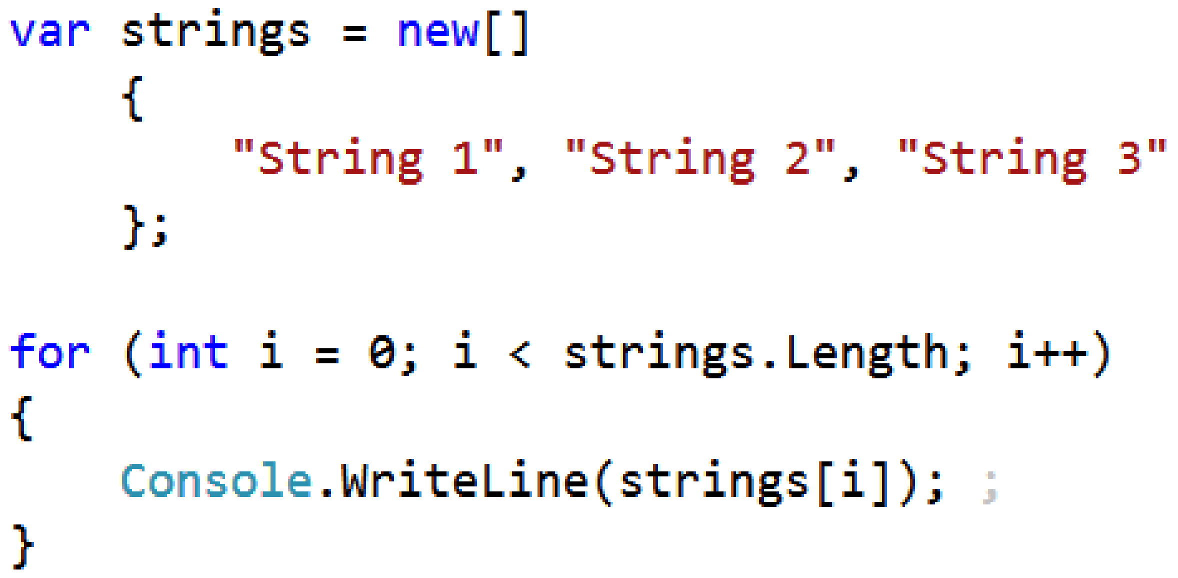
```
var loopCounter = 0;

while (loopCounter > 0)
{
    Console.WriteLine("This will not execute!");
}

do
{
    Console.WriteLine("This will execute once!");
} while (loopCounter > 0);
```

Iteration Statements: for

- *for loops* are similar to *while loops*
- *for loops* include clauses that execute before the loop begins and after every iteration:
  - Initialization clause – typically used to initialize one of more loop variables
  - Iteration clause – typically used to update the loop variable

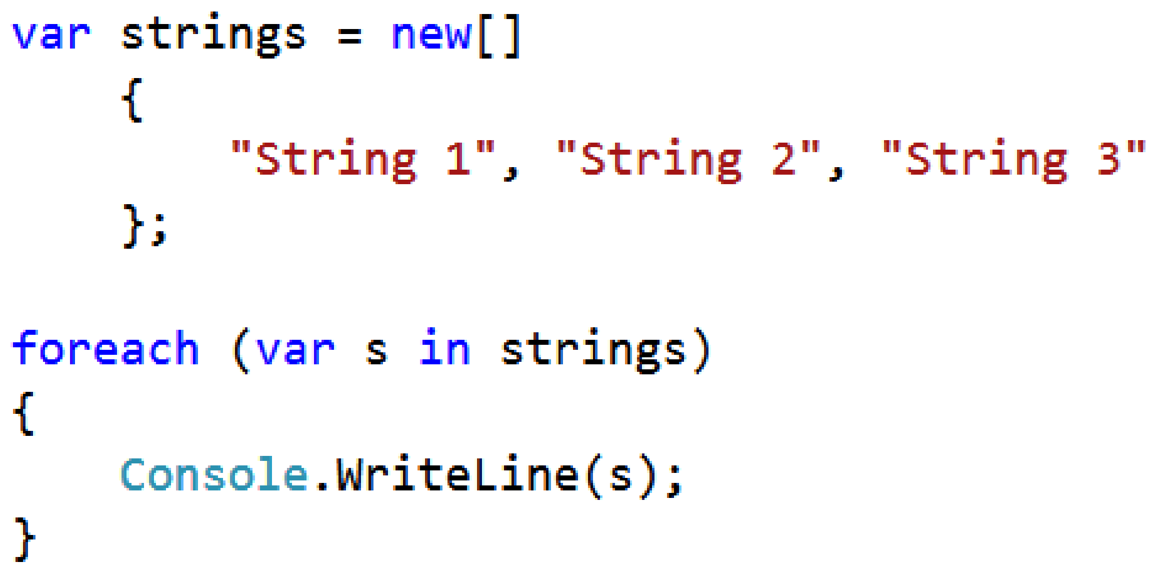


```
var strings = new[]
    {
        "String 1", "String 2", "String 3"
    };

for (int i = 0; i < strings.Length; i++)
{
    Console.WriteLine(strings[i]); ;
}
```

Iteration Statements: foreach

- *Foreach loop* iterates over each element in an enumerable object
  - Array, Collection, List<T>



```
var strings = new[]
    {
        "String 1", "String 2", "String 3"
    };

foreach (var s in strings)
{
    Console.WriteLine(s);
}
```

Jump Statements

- Jump statements redirect execution
  - *break* ends a loop or exits a switch
  - *continue* skips a loop iteration and starts the next iteration
  - *goto* transfers execution to a position marked by a label
  - *return* exits a method
  - *throw* raises an exception

DEMO

Microsoft

Iteration, break and continue (010)

Manipulating Types

MVA Microsoft Virtual Academy

Microsoft

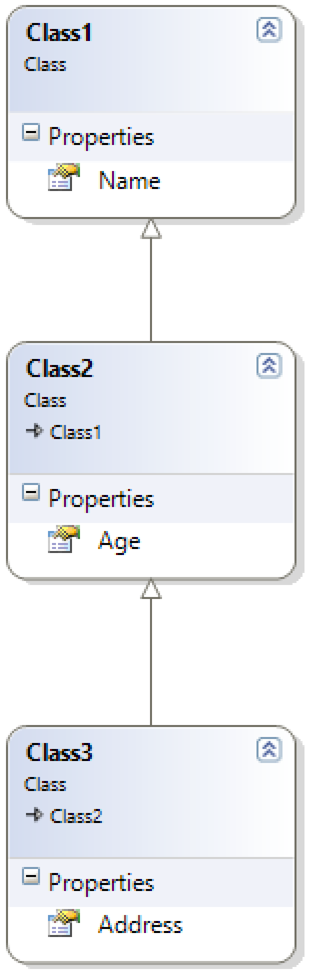
Casting Types

- Casting allows us to work with types in a general sense – as their base object or as an instance of an interface implementation.
- We can explicitly attempt to cast an object to another type
  - An advantage of strong typing is that the compiler often knows when a cast is possible. It doesn't always know.
  - Compilation will fail if the compiler detects an invalid cast.
- But, what about scenarios that the compiler can't detect?

Casting Types

- In this scenario, instances of Class2 and Class3 can be cast an instance of Class1.
- However, an instance of Class2 can never be cast to Class3.
- If Class2 is cast to Class1, can it then be cast to Class3?

```
public class Class1
{
    public string Name { get; set; }
}
public class Class2 : Class1
{
    public int Age { get; set; }
}
public class Class3 : Class2
{
    public string Address { get; set; }
}
```



Is and As

- C# provides us with the *is* operator
  - returns true if an object *is* an instance of a type
- The *as* operator attempts to cast an object to a specified type
  - returning the instance cast to the type
  - null if not possible
  - does not raise an exception

DEMO

Microsoft

Casting Demo

Manipulating Strings

MVA Microsoft Virtual Academy

Microsoft

What is a string?

- A string object is an immutable (unchangeable) sequence of characters.
- Any method that manipulates a string, actually returns a *new* string.
- The StringBuilder class provides a mutable implementation of a string.

DEMO

Microsoft

StringBuilder (012)

String Manipulation

- The string class provides many methods for manipulating strings
- Bear in mind that *new* string objects are returned for:
  - Replace
  - ToUpper
  - Concat

```
var source = "The quick brown fox jumped over the lazy dog";

var value = source.Substring(4, 5);
// value = "quick"

value = string.Concat(value, " fox");
// value == "quick fox"

value = value.Replace("fox", "dog");
// value == "quick dog"

value = value.ToUpper();
// value == "QUICK DOG"

var array = "dog".ToArray();
// array == { 'd', 'o', 'g' }

var bytes = Encoding.ASCII.GetBytes("dog");
// bytes = byte[] of "dog"
```

Regular Expressions

- Regular expressions are a specialized syntax to find and replace patterns in strings

```
var source = "The quick brown fox jumped over the lazy dog";

var split = source.Split(' ');
var value = split[1];
// value = "quick"

var pattern = @"\b\w+\b";
var matches = Regex.Matches(source, pattern);
value = matches[1].Value;
// value = "quick"

pattern = "(The )(.+)( brown)";
var groups = Regex.Match(source, pattern).Groups;
// group0 = "The quick brown"
// group1 = "The "
// group2 = "quick"
// group3 = " brown"
```

Module Recap

- Controlling Programmatic Flow
- Manipulating Types and Strings

Microsoft