VB.NET and C# Comparison

This is a quick reference guide to highlight some key syntactical differences between VB.NET and C#. Hope you find this useful!
Thank you to Tom Shelton, Fergus Cooney, Steven Swafford, Gjuro Kladaric, and others for your contributions.
Also see Java and C# Comparison.

Program Structure

VB.NET

```
Imports System

Namespace Hello
  Class HelloWorld
    Overloads Shared Sub Main(ByVal args() As String)
      Dim name As String = "VB.NET"

      'See if an argument was passed from the command line
      If args.Length = 1 Then name = args(0)

      Console.WriteLine("Hello, " & name & "!")
    End Sub
  End Class
End Namespace
```

C#

```
using System;

namespace Hello {
  public class HelloWorld {
    public static void Main(string[] args) {
      string name = "C#";

      // See if an argument was passed from the command line
      if (args.Length == 1)
        name = args[0];

      Console.WriteLine("Hello, " + name + "!");
    }
  }
}
```

Comments

VB.NET

```
' Single line only

REM Single line only

''' <summary>XML comments</summary>
```

C#

```
// Single line
/* Multiple
   line */
/// <summary>XML comments on single line</summary>
/** <summary>XML comments on multiple lines</summary> */
```

Data Types

VB.NET

```
Value Types
Boolean
Byte, SByte
Char
Short, UShort, Integer, UInteger, Long, ULong
Single, Double
Decimal
Date   (alias of System.DateTime)
structures
enumerations

Reference Types
objects
String
arrays
delegates

Initializing
Dim correct As Boolean = True
Dim b As Byte = &H2A   'hex or &O52 for octal
Dim person As Object = Nothing
Dim name As String = "Dwight"
Dim grade As Char = "B"c
Dim today As Date = #12/31/2010 12:15:00 PM#
Dim amount As Decimal = 35.99@
Dim gpa As Single = 2.9!
Dim pi As Double = 3.14159265
Dim lTotal As Long = 123456L
Dim sTotal As Short = 123S
Dim usTotal As UShort = 123US
Dim uiTotal As UInteger = 123UI
Dim ulTotal As ULong = 123UL

Nullable Types
Dim x? As Integer = Nothing

Anonymous Types
Dim stu = New With {.Name = "Sue", .Gpa = 3.4}
Dim stu2 = New With {Key .Name = "Bob", .Gpa = 2.9}

Implicitly Typed Local Variables
Dim s = "Hello!"
Dim nums = New Integer() {1, 2, 3}
Dim hero = New SuperHero With {.Name = "Batman"}

Type Information
Dim x As Integer
Console.WriteLine(x.GetType())          ' System.Int32
Console.WriteLine(GetType(Integer))     ' System.Int32
```

C#

```
Value Types
bool
byte, sbyte
char
short, ushort, int, uint, long, ulong
float, double
decimal
DateTime   (not a built-in C# type)
structs
enumerations

Reference Types
objects
string
arrays
delegates

Initializing
bool correct = true;
byte b = 0x2A;   // hex
object person = null;
string name = "Dwight";
char grade = 'B';
DateTime today = DateTime.Parse("12/31/2010 12:15:00 PM");
decimal amount = 35.99m;
float gpa = 2.9f;
double pi = 3.14159265;   // or 3.14159265D
long lTotal = 123456L;
short sTotal = 123;
ushort usTotal = 123;
uint uiTotal = 123;   // or 123U
ulong ulTotal = 123;   // or 123UL

Nullable Types
int? x = null;

Anonymous Types
var stu = new {Name = "Sue", Gpa = 3.5};
var stu2 = new {Name = "Bob", Gpa = 2.9};   // no Key equivalent

Implicitly Typed Local Variables
var s = "Hello!";
var nums = new int[] { 1, 2, 3 };
var hero = new SuperHero() { Name = "Batman" };

Type Information
int x;
Console.WriteLine(x.GetType());          // System.Int32
Console.WriteLine(typeof(int));           // System.Int32
```

```
Console.WriteLine(TypeName(x))        ' Integer

Dim c as New Circle
isShape = TypeOf c Is Shape    ' True if c is a Shape

isSame = o1 Is o2   // True if o1 and o2 reference same object

Type Conversion / Casting
Dim d As Single = 3.5
Dim i As Integer = CType(d, Integer)   ' set to 4 (Banker's rounding)
i = CInt(d)   ' same result as CType
i = Int(d)    ' set to 3 (Int function truncates the decimal)

Dim s As New Shape
Dim c As Circle = TryCast(s, Circle)   ' Returns Nothing if type cast fails
c = DirectCast(s, Circle)   ' Throws InvalidCastException if type cast fails
```

```
Console.WriteLine(x.GetType().Name);   // Int32

Circle c = new Circle();
isShape = c is Shape;   // true if c is a Shape

isSame = Object.ReferenceEquals(o1, o2)   // true if o1 and o2 reference same object

Type Conversion / Casting
float d = 3.5f;
i = Convert.ToInt32(d);     // Set to 4 (rounds)
int i = (int)d;     // set to 3 (truncates decimal)


Shape s = new Shape();
Circle c = s as Circle;   // Returns null if type cast fails
c = (Circle)s;   // Throws InvalidCastException if type cast fails
```

VB.NET — Constants — C#

```
Const MAX_STUDENTS As Integer = 25

' Can set to a const or var; may be initialized in a constructor
ReadOnly MIN_DIAMETER As Single = 4.93
```

```
const int MAX_STUDENTS = 25;

// Can set to a const or var; may be initialized in a constructor
readonly float MIN_DIAMETER = 4.93f;
```

VB.NET — Enumerations — C#

```
Enum Action
  Start
  [Stop]    ' Stop is a reserved word
  Rewind
  Forward
End Enum

Enum Status
  Flunk = 50
  Pass = 70
  Excel = 90
End Enum

Dim a As Action = Action.Stop
If a <> Action.Start Then _
   Console.WriteLine(a.ToString & " is " & a)      ' "Stop is 1"

Console.WriteLine(Status.Pass)      ' 70
Console.WriteLine(Status.Pass.ToString)      ' Pass
```

```
enum Action {Start, Stop, Rewind, Forward};
enum Status {Flunk = 50, Pass = 70, Excel = 90};

Action a = Action.Stop;
if (a != Action.Start)
  Console.WriteLine(a + " is " + (int) a);    // "Stop is 1"

Console.WriteLine((int) Status.Pass);    // 70
Console.WriteLine(Status.Pass);      // Pass
```

VB.NET — Operators — C#

```
Comparison
=  <  >  <=  >=  <>

Arithmetic
+  -  *  /
Mod
\  (integer division)
^  (raise to a power)

Assignment
=  +=  -=  *=  /=  \=  ^=  <<=  >>=  &=

Bitwise
And   Or   Xor   Not   <<   >>

Logical
AndAlso   OrElse   And   Or   Xor   Not

Note: AndAlso and OrElse perform short-circuit logical evaluations

String Concatenation
&
```

```
Comparison
==  <  >  <=  >=  !=

Arithmetic
+  -  *  /
%  (mod)
/  (integer division if both operands are ints)
Math.Pow(x, y)

Assignment
=  +=  -=  *=  /=  %=  &=  |=  ^=  <<=  >>=  ++  --

Bitwise
&  |  ^  ~  <<  >>

Logical
&&  ||  &  |  ^  !

Note: && and || perform short-circuit logical evaluations

String Concatenation
+
```

VB.NET — Choices — C#

```
' Null-coalescing operator if called with 2 arguments
x = If(y, 5)   ' if y is not Nothing then x = y, else x = 5

' Ternary/Conditional operator (IIf evaluates 2nd and 3rd expressions)
greeting = If(age < 20, "What's up?", "Hello")

' One line doesn't require "End If"
If age < 20 Then greeting = "What's up?"
If age < 20 Then greeting = "What's up?" Else greeting = "Hello"

' Use : to put two commands on same line
If x <> 100 AndAlso y < 5 Then x *= 5 : y *= 2

' Preferred
If x <> 100 AndAlso y < 5 Then
  x *= 5
  y *= 2
End If

' Use _ to break up long single line or use implicit line break
If whenYouHaveAReally < longLine And
  itNeedsToBeBrokenInto2 > Lines Then _
```

```
// Null-coalescing operator
x = y ?? 5;   // if y != null then x = y, else x = 5

// Ternary/Conditional operator
greeting = age < 20 ? "What's up?" : "Hello";

if (age < 20)
  greeting = "What's up?";
else
  greeting = "Hello";

// Multiple statements must be enclosed in {}
if (x != 100 && y < 5) {
  x *= 5;
  y *= 2;
}


No need for _ or : since ; is used to terminate each statement.
```

```
UseTheUnderscore(charToBreakItUp)
```

```
If x > 5 Then
  x *= y
ElseIf x = 5 OrElse y Mod 2 = 0 Then
  x += y
ElseIf x < 10 Then
  x -= y
Else
  x /= y
End If

Select Case color    ' Must be a primitive data type
  Case "pink", "red"
    r += 1
  Case "blue"
    b += 1
  Case "green"
    g += 1
  Case Else
    other += 1
End Select
```

```
if (x > 5)
  x *= y;
else if (x == 5 || y % 2 == 0)
  x += y;
else if (x < 10)
  x -= y;
else
  x /= y;


// Every case must end with break or goto case
switch (color) {                        // Must be integer or string
  case "pink":
  case "red":    r++;    break;
  case "blue":   b++;    break;
  case "green": g++;    break;
  default:    other++;   break;       // break necessary on default
}
```

VB.NET — Loops — C#

Pre-test Loops:

```
While c < 10
  c += 1
End While
```

```
Do Until c = 10
  c += 1
Loop
```

```
Do While c < 10
  c += 1
Loop
```

```
For c = 2 To 10 Step 2
  Console.WriteLine(c)
Next
```

Post-test Loops:

```
Do
  c += 1
Loop While c < 10
```

```
Do
  c += 1
Loop Until c = 10
```

```
' Array or collection looping
Dim names As String() = {"Fred", "Sue", "Barney"}
For Each s As String In names
  Console.WriteLine(s)
Next

' Breaking out of loops
Dim i As Integer = 0
While (True)
  If (i = 5) Then Exit While
  i += 1
End While


' Continue to next iteration
For i = 0 To 4
  If i < 4 Then Continue For
  Console.WriteLine(i)    ' Only prints 4
Next
```

Pre-test Loops:

```
// no "until" keyword
while (c < 10)
  c++;


for (c = 2; c <= 10; c += 2)
  Console.WriteLine(c);
```

Post-test Loop:

```
do
  c++;
while (c < 10);
```

```
// Array or collection looping
string[] names = {"Fred", "Sue", "Barney"};
foreach (string s in names)
  Console.WriteLine(s);


// Breaking out of loops
int i = 0;
while (true) {
  if (i == 5)
    break;
  i++;
}

// Continue to next iteration
for (i = 0; i <= 4; i++) {
  if (i < 4)
    continue;
  Console.WriteLine(i);   // Only prints 4
}
```

VB.NET — Arrays — C#

```
Dim nums() As Integer = {1, 2, 3}
For i As Integer = 0 To nums.Length - 1
  Console.WriteLine(nums(i))
Next

' 4 is the index of the last element, so it holds 5 elements
Dim names(4) As String
names(0) = "David"
names(5) = "Bobby"   ' Throws System.IndexOutOfRangeException

' Resize the array, keeping the existing values (Preserve is optional)
ReDim Preserve names(6)

Dim twoD(rows-1, cols-1) As Single
twoD(2, 0) = 4.5

Dim jagged()() As Integer = { _
  New Integer(4) {}, New Integer(1) {}, New Integer(2) {} }
jagged(0)(4) = 5
```

```
int[] nums = {1, 2, 3};
for (int i = 0; i < nums.Length; i++)
  Console.WriteLine(nums[i]);


// 5 is the size of the array
string[] names = new string[5];
names[0] = "David";
names[5] = "Bobby";   // Throws System.IndexOutOfRangeException

// Add two elements, keeping the existing values
Array.Resize(ref names, 7);

float[,] twoD = new float[rows, cols];
twoD[2,0] = 4.5f;

int[][] jagged = new int[3][] {
  new int[5], new int[2], new int[3] };
jagged[0][4] = 5;
```

VB.NET — Functions — C#

```
' Pass by value (in, default), reference (in/out), and reference (out)
Sub TestFunc(ByVal x As Integer, ByRef y As Integer, ByRef z As Integer)
  x += 1
  y += 1
  z = 5
```

```
// Pass by value (in, default), reference (in/out), and reference (out)
void TestFunc(int x, ref int y, out int z) {
  x++;
  y++;
  z = 5;
```

```vb
End Sub

Dim a = 1, b = 1, c As Integer   ' c set to zero by default
TestFunc(a, b, c)
Console.WriteLine("{0} {1} {2}", a, b, c)   ' 1 2 5

' Accept variable number of arguments
Function Sum(ByVal ParamArray nums As Integer()) As Integer
  Sum = 0
  For Each i As Integer In nums
    Sum += i
  Next
End Function   ' Or use Return statement like C#

Dim total As Integer = Sum(4, 3, 2, 1)   ' returns 10

' Optional parameters must be listed last and must have a default value
Sub SayHello(ByVal name As String, Optional ByVal prefix As String = "")
  Console.WriteLine("Greetings, " & prefix & " " & name)
End Sub

SayHello("Strangelove", "Dr.")
SayHello("Mom")
```

```cs
}

int a = 1, b = 1, c;  // c doesn't need initializing
TestFunc(a, ref b, out c);
Console.WriteLine("{0} {1} {2}", a, b, c);  // 1 2 5

// Accept variable number of arguments
int Sum(params int[] nums) {
  int sum = 0;
  foreach (int i in nums)
    sum += i;
  return sum;
}

int total = Sum(4, 3, 2, 1);   // returns 10

/* C# 4.0 supports optional parameters. Previous versions required function overloading.
*/
void SayHello(string name, string prefix = "") {
  Console.WriteLine("Greetings, " + prefix + " " + name);
}

SayHello("Strangelove", "Dr.");
SayHello("Mom");
```

VB.NET — Strings — C#

```vb
Special character constants (all also accessible from ControlChars class)
vbCrLf, vbCr, vbLf, vbNewLine
vbNullString
vbTab
vbBack
vbFormFeed
vbVerticalTab
""

' String concatenation (use & or +)
Dim school As String = "Harding" & vbTab
school = school & "University"   ' school is "Harding (tab) University"
school &= "University"   ' Same thing (+= does the same)

' Chars
Dim letter As Char = school.Chars(0)   ' letter is H
letter = "Z"c                          ' letter is Z
letter = Convert.ToChar(65)            ' letter is A
letter = Chr(65)                       ' same thing
Dim word() As Char = school.ToCharArray ' word holds Harding

' No string literal operator
Dim filename As String = "c:\temp\x.dat"

' String comparison
Dim mascot As String = "Bisons"
If (mascot = "Bisons") Then    ' true
If (mascot.Equals("Bisons")) Then   ' true
If (mascot.ToUpper().Equals("BISONS")) Then  ' true
If (mascot.CompareTo("Bisons") = 0) Then   ' true

' String matching with Like - Regex is more powerful
If ("John 3:16" Like "Jo[Hh]? #:*") Then   'true

' Substring
s = mascot.Substring(2, 3)) ' son
s = Mid("testing", 2, 3)   ' est

' Replacement
s = mascot.Replace("sons", "nomial")) ' s is "Binomial"

' Split
Dim names As String = "Michael,Dwight,Jim,Pam"
Dim parts() As String = names.Split(",".ToCharArray())   ' One name in each slot

' Date to string
Dim dt As New DateTime(1973, 10, 12)
Dim s As String = "My birthday: " & dt.ToString("MMM dd, yyyy")   ' Oct 12, 1973

' Integer to String
Dim x As Integer = 2
Dim y As String = x.ToString()     ' y is "2"

' String to Integer
Dim x As Integer = Convert.ToInt32("-5")     ' x is -5

' Mutable string
Dim buffer As New System.Text.StringBuilder("two ")
buffer.Append("three ")
buffer.Insert(0, "one ")
buffer.Replace("two", "TWO")
Console.WriteLine(buffer)          ' Prints "one TWO three"
```

```cs
Escape sequences
\r    // carriage-return
\n    // line-feed
\t    // tab
\\    // backslash
\"    // quote

// String concatenation
string school = "Harding\t";
school = school + "University";   // school is "Harding (tab) University"
school += "University";   // Same thing

// Chars
char letter = school[0];            // letter is H
letter = 'Z';                       // letter is Z
letter = Convert.ToChar(65);     // letter is A
letter = (char)65;                  // same thing
char[] word = school.ToCharArray();   // word holds Harding

// String literal
string filename = @"c:\temp\x.dat";   // Same as "c:||temp||x.dat"

// String comparison
string mascot = "Bisons";
if (mascot == "Bisons")    // true
if (mascot.Equals("Bisons"))   // true
if (mascot.ToUpper().Equals("BISONS"))   // true
if (mascot.CompareTo("Bisons") == 0)    // true

// String matching - No Like equivalent, use Regex


// Substring
s = mascot.Substring(2, 3))     // son
s = "testing".Substring(1, 3);   // est (no Mid)

// Replacement
s = mascot.Replace("sons", "nomial"))     // Binomial

// Split
string names = "Michael,Dwight,Jim,Pam";
string[] parts = names.Split(",".ToCharArray());   // One name in each slot

// Date to string
DateTime dt = new DateTime(1973, 10, 12);
string s = dt.ToString("MMM dd, yyyy");     // Oct 12, 1973

// int to string
int x = 2;
string y = x.ToString();     // y is "2"

// string to int
int x = Convert.ToInt32("-5");     // x is -5

// Mutable string
System.Text.StringBuilder buffer = new System.Text.StringBuilder("two ");
buffer.Append("three ");
buffer.Insert(0, "one ");
buffer.Replace("two", "TWO");
Console.WriteLine(buffer);     // Prints "one TWO three"
```

VB.NET — Regular Expressions — C#

```vb
Imports System.Text.RegularExpressions

' Match a string pattern
Dim r As New Regex("j[aeiou]h?. \d:*", RegexOptions.IgnoreCase Or _
      RegexOptions.Compiled)
If (r.Match("John 3:16").Success) Then   'true
   Console.WriteLine("Match")
End If

' Find and remember all matching patterns
Dim s As String = "My number is 305-1881, not 305-1818."
Dim r As New Regex("(\d+-\d+)")
```

```cs
using System.Text.RegularExpressions;

// Match a string pattern
Regex r = new Regex(@"j[aeiou]h?. \d:*", RegexOptions.IgnoreCase |
      RegexOptions.Compiled);
if (r.Match("John 3:16").Success)   // true
   Console.WriteLine("Match");

// Find and remember all matching patterns
string s = "My number is 305-1881, not 305-1818.";
Regex r = new Regex("(\\d+-\\d+)");
```

```vb
Dim m As Match = r.Match(s)      ' Matches 305-1881 and 305-1818
While m.Success
    Console.WriteLine("Found number: " & m.Groups(1).Value & " at position " _
          & m.Groups(1).Index.ToString)
    m = m.NextMatch()
End While

' Remeber multiple parts of matched pattern
Dim r As New Regex("(\d\d):(\d\d) (am|pm)")
Dim m As Match = r.Match("We left at 03:15 pm.")
If m.Success Then
    Console.WriteLine("Hour: " & m.Groups(1).ToString)        ' 03
    Console.WriteLine("Min: " & m.Groups(2).ToString)         ' 15
    Console.WriteLine("Ending: " & m.Groups(3).ToString)    ' pm
End If

' Replace all occurrances of a pattern
Dim r As New Regex("h\w+?d", RegexOptions.IgnoreCase)
Dim s As String = r.Replace("I heard this was HARD!", "easy")    ' I easy this was easy!

' Replace matched patterns
Dim s As String = Regex.Replace("123 < 456", "(\d+) . (\d+)", "$2 > $1")    ' 456 > 123

' Split a string based on a pattern
Dim names As String = "Michael, Dwight, Jim, Pam"
Dim r As New Regex(",\s*")
Dim parts() As String = r.Split(names)    ' One name in each slot
```

```cs
// Matches 305-1881 and 305-1818
for (Match m = r.Match(s); m.Success; m = m.NextMatch())
    Console.WriteLine("Found number: " + m.Groups[1] + " at position " +
        m.Groups[1].Index);

// Remeber multiple parts of matched pattern
Regex r = new Regex("@(\d\d):(\d\d) (am|pm)");
Match m = r.Match("We left at 03:15 pm.");
if (m.Success) {
    Console.WriteLine("Hour: " + m.Groups[1]);       // 03
    Console.WriteLine("Min: " + m.Groups[2]);        // 15
    Console.WriteLine("Ending: " + m.Groups[3]);   // pm
}

// Replace all occurrances of a pattern
Regex r = new Regex("h\\w+?d", RegexOptions.IgnoreCase);
string s = r.Replace("I heard this was HARD!", "easy"));   // I easy this was easy!

// Replace matched patterns
string s = Regex.Replace("123 < 456", @"(\d+) . (\d+)", "$2 > $1");   // 456 > 123

// Split a string based on a pattern
string names = "Michael, Dwight, Jim, Pam";
Regex r = new Regex(@",\s*");
string[] parts = r.Split(names);   // One name in each slot
```

VB.NET — Exception Handling — C#

```vb
' Throw an exception
Dim ex New Exception("Something is really wrong.")
Throw  ex

' Catch an exception
Try
  y = 0
  x = 10 / y
Catch ex As Exception When y = 0  ' Argument and When is optional
  Console.WriteLine(ex.Message)
Finally
  Beep()
End Try

' Deprecated unstructured error handling
On Error GoTo MyErrorHandler
...
MyErrorHandler: Console.WriteLine(Err.Description)
```

```cs
// Throw an exception
Exception up = new Exception("Something is really wrong.");
throw up;  // ha ha

// Catch an exception
try {
  y = 0;
  x = 10 / y;
}
catch (Exception ex) {   // Argument is optional, no "When" keyword
  Console.WriteLine(ex.Message);
}
finally {
  Microsoft.VisualBasic.Interaction.Beep();
}
```

VB.NET — Namespaces — C#

```vb
Namespace Harding.Compsci.Graphics
  ...
End Namespace

' or

Namespace Harding
  Namespace Compsci
    Namespace Graphics
      ...
    End Namespace
  End Namespace
End Namespace

Imports Harding.Compsci.Graphics
```

```cs
namespace Harding.Compsci.Graphics {
  ...
}

// or

namespace Harding {
  namespace Compsci {
    namespace Graphics {
      ...
    }
  }
}

using Harding.Compsci.Graphics;
```

VB.NET — Classes & Interfaces — C#

```vb
' Access Modifiers
Public
Private
Friend
Protected
Protected Friend

' Class Modifiers
MustInherit
NotInheritable


' Method Modifiers
MustOverride
NotInheritable
Shared
Overridable

' All members are Shared
Module

' Partial classes
Partial Class Team
  ...
  Protected name As String
  Public Overridable Sub DisplayName()
    Console.WriteLine(name)
  End Sub
```

```cs
// Access Modifiers
public
private
internal
protected
protected internal

// Class Modifiers
abstract
sealed
static

// Method Modifiers
abstract
sealed
static
virtual

// No Module equivalent - just use static class


// Partial classes
partial class Team {
  ...
  protected string name;
  public virtual void DislpayName() {
    Console.WriteLine(name);
}
```

```
End Class

' Inheritance
Class FootballTeam
  Inherits Team
  ...
  Public Overrides Sub DisplayName()
    Console.WriteLine("** " + name + " **")
  End Sub
End Class

' Interface definition
Interface IAlarmClock
  Sub Ring()
  Property TriggerDateTime() As DateTime
End Interface

' Extending an interface
Interface IAlarmClock
  Inherits IClock
  ...
End Interface

' Interface implementation
Class WristWatch
  Implements IAlarmClock, ITimer

  Public Sub Ring() Implements IAlarmClock.Ring
    Console.WriteLine("Wake up!")
  End Sub

  Public Property TriggerDateTime As DateTime Implements
IAlarmClock.TriggerDateTime
  ...
End Class
```

```
// Inheritance
class FootballTeam : Team {
  ...
  public override void DislpayName() {
    Console.WriteLine("** " + name + " **");
  }
}

// Interface definition
interface IAlarmClock {
  void Ring();
  DateTime CurrentDateTime { get; set; }
}

// Extending an interface
interface IAlarmClock : IClock {
  ...
}

// Interface implementation
class WristWatch : IAlarmClock, ITimer {

  public void Ring() {
    Console.WriteLine("Wake up!");
  }

  public DateTime TriggerDateTime { get; set; }
  ...
}
```

VB.NET — Constructors & Destructors — C#

```
Class SuperHero
  Inherits Person

  Private powerLevel As Integer
  Private name As String

  ' Default constructor
  Public Sub New()
    powerLevel = 0
    name = "Super Bison"
  End Sub

  Public Sub New(ByVal powerLevel As Integer)
    Me.New("Super Bison")    ' Call other constructor
    Me.powerLevel = powerLevel
  End Sub

  Public Sub New(ByVal name As String)
    MyBase.New(name)    ' Call base classes' constructor
    Me.name = name
  End Sub

  Shared Sub New()
    ' Shared constructor invoked before 1st instance is created
  End Sub

  Protected Overrides Sub Finalize()
    ' Destructor to free unmanaged resources
    MyBase.Finalize()
  End Sub
End Class
```

```
class SuperHero : Person {

  private int powerLevel;
  private string name;


  // Default constructor
  public SuperHero() {
    powerLevel = 0;
    name = "Super Bison";
  }

  public SuperHero(int powerLevel)
    : this("Super Bison") {   // Call other constructor
    this.powerLevel = powerLevel;
  }

  public SuperHero(string name)
    : base(name) {   // Call base classes' constructor
    this.name = name;
  }

  static SuperHero() {
    // Static constructor invoked before 1st instance is created
  }

  ~SuperHero() {
    // Destructor implicitly creates a Finalize method
  }

}
```

VB.NET — Using Objects — C#

```
Dim hero As SuperHero = New SuperHero
' or
Dim hero As New SuperHero

With hero
  .Name = "SpamMan"
  .PowerLevel = 3
End With

hero.Defend("Laura Jones")
hero.Rest()     ' Calling Shared method
' or
SuperHero.Rest()

Dim hero2 As SuperHero = hero   ' Both reference the same object
hero2.Name = "WormWoman"
Console.WriteLine(hero.Name)    ' Prints WormWoman

hero = Nothing     ' Free the object

If hero Is Nothing Then _
  hero = New SuperHero

Dim obj As Object = New SuperHero
If TypeOf obj Is SuperHero Then _
  Console.WriteLine("Is a SuperHero object.")

' Mark object for quick disposal
Using reader As StreamReader = File.OpenText("test.txt")
  Dim line As String = reader.ReadLine()
```

```
SuperHero hero = new SuperHero();


// No "With" but can use object initializers
SuperHero hero = new SuperHero() { Name = "SpamMan", PowerLevel = 3 };

hero.Defend("Laura Jones");
SuperHero.Rest();   // Calling static method

SuperHero hero2 = hero;   // Both reference the same object
hero2.Name = "WormWoman";
Console.WriteLine(hero.Name);   // Prints WormWoman

hero = null ;   // Free the object

if (hero == null)
  hero = new SuperHero();

Object obj = new SuperHero();
if (obj is SuperHero)
  Console.WriteLine("Is a SuperHero object.");

// Mark object for quick disposal
using (StreamReader reader = File.OpenText("test.txt")) {
  string line;
```

```
  While Not line Is Nothing
    Console.WriteLine(line)
    line = reader.ReadLine()
  End While
End Using
```

```
  while ((line = reader.ReadLine()) != null)
    Console.WriteLine(line);
}
```

VB.NET Structs C#

```
Structure Student
  Public name As String
  Public gpa As Single

  Public Sub New(ByVal name As String, ByVal gpa As Single)
    Me.name = name
    Me.gpa = gpa
  End Sub
End Structure

Dim stu As Student = New Student("Bob", 3.5)
Dim stu2 As Student = stu

stu2.name = "Sue"
Console.WriteLine(stu.name)     ' Prints Bob
Console.WriteLine(stu2.name)   ' Prints Sue
```

```
struct Student {
  public string name;
  public float gpa;

  public Student(string name, float gpa) {
    this.name = name;
    this.gpa = gpa;
  }
}

Student stu = new Student("Bob", 3.5f);
Student stu2 = stu;

stu2.name = "Sue";
Console.WriteLine(stu.name);    // Prints Bob
Console.WriteLine(stu2.name);   // Prints Sue
```

VB.NET Properties C#

```
' Auto-implemented properties are new to VB10
Public Property Name As String
Public Property Size As Integer = -1      ' Default value, Get and Set both Public

' Traditional property implementation
Private mName As String
Public Property Name() As String
    Get
        Return mName
    End Get
    Set(ByVal value As String)
        mName = value
    End Set
End Property

' Read-only property
Private mPowerLevel As Integer
Public ReadOnly Property PowerLevel() As Integer
    Get
        Return mPowerLevel
    End Get
End Property

' Write-only property
Private mHeight As Double
Public WriteOnly Property Height() As Double
    Set(ByVal value As Double)
        mHeight = If(value < 0, mHeight = 0, mHeight = value)
    End Set
End Property
```

```
// Auto-implemented properties
public string Name { get; set; }
public int Size { get; protected set; }     // Set default value in constructor

// Traditional property implementation
private string name;
public string Name {
  get {
    return name;
  }
  set {
    name = value;
  }
}

// Read-only property
private int powerLevel;
public int PowerLevel {
  get {
    return powerLevel;
  }
}

// Write-only property
private double height;
public double Height {
  set {
    height = value < 0 ? 0 : value;
  }
}
```

VB.NET Generics C#

```vb
' Enforce accepted data type at compile-time
Dim numbers As New List(Of Integer)
numbers.Add(2)
numbers.Add(4)
DisplayList(Of Integer)(numbers)

' Subroutine can display any type of List
Sub DisplayList(Of T)(ByVal list As List(Of T))
    For Each item As T In list
        Console.WriteLine(item)
    Next
End Sub

' Class works on any data type
Class SillyList(Of T)
    Private list(10) As T
    Private rand As New Random

    Public Sub Add(ByVal item As T)
        list(rand.Next(10)) = item
    End Sub

    Public Function GetItem() As T
        Return list(rand.Next(10))
    End Function
End Class

' Limit T to only types that implement IComparable
Function Maximum(Of T As IComparable)(ByVal ParamArray items As T()) As T
    Dim max As T = items(0)
    For Each item As T In items
        If item.CompareTo(max) > 0 Then max = item
    Next
    Return max
End Function
```

```cs
// Enforce accepted data type at compile-time
List<int> numbers = new List<int>();
numbers.Add(2);
numbers.Add(4);
DisplayList<int>(numbers);

// Function can display any type of List
void DisplayList<T>(List<T> list) {
    foreach (T item in list)
        Console.WriteLine(item);
}

// Class works on any data type
class SillyList<T> {
    private T[] list = new T[10];
    private Random rand = new Random();

    public void Add(T item) {
        list[rand.Next(10)] = item;
    }

    public T GetItem() {
        return list[rand.Next(10)];
    }
}

// Limit T to only types that implement IComparable
T Maximum<T>(params T[] items) where T : IComparable<T> {
    T max = items[0];
    foreach (T item in items)
        if (item.CompareTo(max) > 0)
            max = item;
    return max;
}
```

VB.NET — Delegates & Lambda Expressions — C#

```vb
Delegate Sub HelloDelegate(ByVal s As String)

Sub SayHello(ByVal s As String)
  Console.WriteLine("Hello, " & s)
End Sub

' Create delegate that calls SayHello
Dim hello As HelloDelegate = AddressOf SayHello
hello("World")   ' Or hello.Invoke("World")

' Use lambda expression (anonymous method) instead of a delegate
Dim hello2 = Sub(x) Console.WriteLine("Hello, " & x)
hello2("World")
```

```cs
delegate void HelloDelegate(string s);

void SayHello(string s) {
  Console.WriteLine("Hello, " + s);
}

// C# 1.0 delegate syntax with named method
HelloDelegate hello = new HelloDelegate(SayHello);
hello("World");   // Or hello.Invoke("World");

// C# 2.0 delegate syntax with anonymous method
HelloDelegate hello2 = delegate(string s) {
  Console.WriteLine("Hello, " + s);
};
hello2("World");

// C# 3.0 delegate syntax with lambda expression
HelloDelegate hello3 = s => { Console.WriteLine("Hello, " + s); };
hello3("World");
```

```vb
' Use Func(Of T, TResult) delegate to call Uppercase
Dim convert As Func(Of String, String) = AddressOf Uppercase
Console.WriteLine(convert("test"))

Function Uppercase(s As String) As String
  Return s.ToUpper
End Function

' Declare and invoke lambda expression
Console.WriteLine((Function(num As Integer) num + 1)(2))

' Pass lambda expression as an argument
TestValues(Function(x, y) x Mod y = 0)

Sub TestValues(ByVal f As Func(Of Integer, Integer, Boolean))
  If f(8, 4) Then
    Console.WriteLine("true")
  Else
    Console.WriteLine("false")
  End If
End Sub
```

```cs
// Use Func<in T, out TResult> delegate to call Uppercase
Func<string, string> convert = Uppercase;
Console.WriteLine(convert("test"));

string Uppercase(string s) {
  return s.ToUpper();
}

// Declare and invoke Func using a lambda expression
Console.WriteLine(new Func<int, int>(num => num + 1)(2));

// Pass lamba expression as an argument
TestValues((x, y) => x % y == 0);

void TestValues(Func<int, int, bool> f) {
  if (f(8, 4))
    Console.WriteLine("true");
  else
    Console.WriteLine("false");
}
```

VB.NET — Extension Methods — C#

```vb
Imports System.Runtime.CompilerServices

Module StringExtensions
  <Extension()>
  Public Function VowelCount(ByVal s As String) As Integer
    Return s.Count(Function(c) "aeiou".Contains(Char.ToLower(c)))
  End Function
End Module

' Using the extension method
Console.WriteLine("This is a test".VowelCount)
```

```cs
public static class StringExtensions {
  public static int VowelCount(this string s) {
    return s.Count(c => "aeiou".Contains(Char.ToLower(c)));
  }
}

// Using the extension method
Console.WriteLine("This is a test".VowelCount());
```

VB.NET — Events — C#

```vb
Delegate Sub MsgArrivedEventHandler(ByVal message As String)

Event MsgArrivedEvent As MsgArrivedEventHandler

' or to define an event which declares a delegate implicitly
Event MsgArrivedEvent(ByVal message As String)

AddHandler MsgArrivedEvent, AddressOf My_MsgArrivedCallback
' Won't throw an exception if obj is Nothing
RaiseEvent MsgArrivedEvent("Test message")
RemoveHandler MsgArrivedEvent, AddressOf My_MsgArrivedCallback

Imports System.Windows.Forms

Dim WithEvents MyButton As Button    ' WithEvents can't be used on local variable
MyButton = New Button

Sub MyButton_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyButton.Click
  MessageBox.Show(Me, "Button was clicked", "Info", _
    MessageBoxButtons.OK, MessageBoxIcon.Information)
End Sub
```

```cs
delegate void MsgArrivedEventHandler(string message);

event MsgArrivedEventHandler MsgArrivedEvent;

// Delegates must be used with events in C#


MsgArrivedEvent += new MsgArrivedEventHandler(My_MsgArrivedEventCallback);
MsgArrivedEvent("Test message");    // Throws exception if obj is null
MsgArrivedEvent -= new MsgArrivedEventHandler(My_MsgArrivedEventCallback);



using System.Windows.Forms;

Button MyButton = new Button();
MyButton.Click += new System.EventHandler(MyButton_Click);

void MyButton_Click(object sender, System.EventArgs e) {
  MessageBox.Show(this, "Button was clicked", "Info",
    MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```

VB.NET — LINQ — C#

```vb
Dim nums() As Integer = {5, 8, 2, 1, 6}

' Get all numbers in the array above 4
Dim results = From n In nums
              Where n > 4
              Select n

' Same thing using lamba expression
results = nums.Where(Function(n) n > 4)

' Displays 5 8 6
For Each n As Integer In results
    Console.Write(n & " ")
Next

Console.WriteLine(results.Count())     ' 3
Console.WriteLine(results.First())     ' 5
Console.WriteLine(results.Last())      ' 6
Console.WriteLine(results.Average())   ' 6.33333

results = results.Intersect({5, 6, 7})     ' 5 6
results = results.Concat({5, 1, 5})     ' 5 6 5 1 5
results = results.Distinct()     ' 5 6 1

Dim Students() As Student = {
    New Student With {.Name = "Bob", .Gpa = 3.5},
    New Student With {.Name = "Sue", .Gpa = 4.0},
    New Student With {.Name = "Joe", .Gpa = 1.9}
}

' Get a list of students ordered by Gpa with Gpa >= 3.0
Dim goodStudents = From s In Students
          Where s.Gpa >= 3.0
          Order By s.Gpa Descending
          Select s

Console.WriteLine(goodStudents.First.Name)     ' Sue
```

```cs
int[] nums = { 5, 8, 2, 1, 6 };

// Get all numbers in the array above 4
var results = from n in nums
              where n > 4
              select n;

// Same thing using lamba expression
results = nums.Where(n => n > 4);

// Displays 5 8 6
foreach (int n in results)
    Console.Write(n + " ");


Console.WriteLine(results.Count());     // 3
Console.WriteLine(results.First());     // 5
Console.WriteLine(results.Last());     // 6
Console.WriteLine(results.Average());     // 6.33333

results = results.Intersect(new[] {5, 6, 7});     // 5 6
results = results.Concat(new[] {5, 1, 5});     // 5 6 5 1 5
results = results.Distinct();     // 5 6 1

Student[] Students = {
    new Student{ Name = "Bob", Gpa = 3.5 },
    new Student{ Name = "Sue", Gpa = 4.0 },
    new Student{ Name = "Joe", Gpa = 1.9 }
};

// Get a list of students ordered by Gpa with Gpa >= 3.0
var goodStudents = from s in Students
          where s.Gpa >= 3.0
          orderby s.Gpa descending
          select s;

Console.WriteLine(goodStudents.First().Name);     // Sue
```

VB.NET — Collections — C#

```vb
Popular classes in System.Collections (stored as Object)
ArrayList
Hashtable
Queue
Stack

Popular classes in System.Collections.Generic (stored as type T)
List(Of T)
SortedList(Of TKey, TValue)
Dictionary(Of TKey, TValue)
Queue(Of T)
Stack(Of T)

Popular classes in System.Collections.Concurrent (thread safe)
BlockingCollection(Of T)
ConcurrentDictionary(Of TKey, TValue)
ConcurrentQueue(Of T)
ConcurrentStack(Of T)

Microsoft.VisualBasic (not recommended)
Collection


' Store ID and name
Dim students As New Dictionary(Of Integer, String) From
{
  {123, "Bob"},
  {444, "Sue"},
  {555, "Jane"}
}

students.Add(987, "Gary")
Console.WriteLine(students(444))    ' Sue

' Display all
For Each stu In students
```

```cs
Popular classes in System.Collections (stored as Object)
ArrayList
Hashtable
Queue
Stack

Popular classes in System.Collections.Generic (stored as type T)
List<T>
SortedList<TKey, TValue>
Dictionary<TKey, TValue>
Queue<T>
Stack<T>

Popular classes in System.Collections.Concurrent (thread safe)
BlockingCollection<T>
ConcurrentDictionary<TKey, TValue>
ConcurrentQueue<T>
ConcurrentStack<T>

No equivalent to Microsoft.VisualBasic.Collection


// Store ID and name
var students = new Dictionary<int, string>
{
  { 123, "Bob" },
  { 444, "Sue" },
  { 555, "Jane" }
};

students.Add(987, "Gary");
Console.WriteLine(students[444]);   // Sue

// Display all
foreach (var stu in students) {
  Console.WriteLine(stu.Key + " = " + stu.Value);
```

```
  Console.WriteLine(stu.Key & " = " & stu.Value)
Next

' Method iterator for custom iteration over a collection
Iterator Function OddNumbers(ByVal lastNum As Integer) As
System.Collections.IEnumerable
  For num = 1 To lastNum
    If num Mod 2 = 1 Then
      Yield num
    End If
  Next
End Function

' 1 3 5 7
For Each num In OddNumbers(7)
  Console.Write(num & " ")
Next
```

```
}

// Method iterator for custom iteration over a collection
static System.Collections.Generic.IEnumerable<int> OddNumbers(int lastNum)
{
  for (var num = 1; num <= lastNum; num++)
    if (num % 2 == 1)
      yield return num;
}

// 1 3 5 7
foreach (double num in OddNumbers(7))
{
  Console.Write(num + " ");
}
```

VB.NET Attributes C#

```
' Attribute can be applied to anything
Public Class IsTestedAttribute
    Inherits Attribute
End Class

' Attribute can only be applied to classes or structs
<AttributeUsage(AttributeTargets.Class Or AttributeTargets.Struct)>
Public Class AuthorAttribute
    Inherits Attribute

    Public Property Name As String
    Public Property Version As Integer = 0

    Public Sub New(ByVal name As String)
        Me.Name = name
    End Sub
End Class

<Author("Sue", Version:=3)>
Class Shape

    <IsTested()>
    Sub Move()
        ' Do something...
    End Sub
End Class
```

```
// Attribute can be applied to anything
public class IsTestedAttribute : Attribute
{
}

// Attribute can only be applied to classes or structs
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Struct)]
public class AuthorAttribute : Attribute {

    public string Name { get; set; }
    public int Version { get; set; }

    public AuthorAttribute(string name) {
        Name = name;
        Version = 0;
    }
}

[Author("Sue", Version = 3)]
class Shape {

    [IsTested]
    void Move() {
        // Do something...
    }
}
```

VB.NET Console I/O C#

```
Console.Write("What's your name? ")
Dim name As String = Console.ReadLine()
Console.Write("How old are you? ")
Dim age As Integer = Val(Console.ReadLine())
Console.WriteLine("{0} is {1} years old.", name, age)
' or
Console.WriteLine(name & " is " & age & " years old.")

Dim c As Integer
c = Console.Read()    ' Read single char
Console.WriteLine(c)  ' Prints 65 if user enters "A"
```

```
Console.Write("What's your name? ");
string name = Console.ReadLine();
Console.Write("How old are you? ");
int age = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("{0} is {1} years old.", name, age);
// or
Console.WriteLine(name + " is " + age + " years old.");


int c = Console.Read();  // Read single char
Console.WriteLine(c);    // Prints 65 if user enters "A"
```

VB.NET File I/O C#

```
Imports System.IO

' Write out to text file
Dim writer As StreamWriter = File.CreateText("c:\myfile.txt")
writer.WriteLine("Out to file.")
writer.Close()

' Read all lines from text file
Dim reader As StreamReader = File.OpenText("c:\myfile.txt")
Dim line As String = reader.ReadLine()
While Not line Is Nothing
  Console.WriteLine(line)
  line = reader.ReadLine()
End While
reader.Close()

' Write out to binary file
Dim str As String = "Text data"
Dim num As Integer = 123
Dim binWriter As New BinaryWriter(File.OpenWrite("c:\myfile.dat"))
binWriter.Write(str)
binWriter.Write(num)
binWriter.Close()

' Read from binary file
Dim binReader As New BinaryReader(File.OpenRead("c:\myfile.dat"))
str = binReader.ReadString()
num = binReader.ReadInt32()
binReader.Close()
```

```
using System.IO;

// Write out to text file
StreamWriter writer = File.CreateText("c:\\myfile.txt");
writer.WriteLine("Out to file.");
writer.Close();

// Read all lines from text file
StreamReader reader = File.OpenText("c:\\myfile.txt");
string line = reader.ReadLine();
while (line != null) {
  Console.WriteLine(line);
  line = reader.ReadLine();
}
reader.Close();

// Write out to binary file
string str = "Text data";
int num = 123;
BinaryWriter binWriter = new BinaryWriter(File.OpenWrite("c:\\myfile.dat"));
binWriter.Write(str);
binWriter.Write(num);
binWriter.Close();

// Read from binary file
BinaryReader binReader = new BinaryReader(File.OpenRead("c:\\myfile.dat"));
str = binReader.ReadString();
num = binReader.ReadInt32();
binReader.Close();
```

Last modified: April 4, 2016

Produced by Frank McCown, Harding University Computer Science Dept
Please send any corrections or comments to fmccown@harding.edu.

Home