

Arrays

C#	COBOL	VB.NET	Java
<pre>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 // C# can't dynamically resize an array. Just copy into new array. string[] names2 = new string[7]; Array.Copy(names, names2, names.Length); // or names.CopyTo(names2, 0); float[,] twoD = new float[rows, cols]; twoD[2,0] = 4.5f; int[][] jagged = new int[][] { new int[] {1, 2}, new int[] {3, 4, 5}, new int[] {6, 7, 8, 9} }; jagged[0][4] = 5;</pre>	<pre>declare nums = table of binary-long (1 2 3) declare names as string occurs 5 *> Can also do: declare names-again as string occurs any set size of names to 5 set names(1) to "David" *> first element indexed as 1 *> ...but can also use zero based subscripting: set names(0) to "David" *> first element indexed as 0 *>set names(6) to "Bobby" *> throws System.IndexOutOfRangeException *> COBOL does not have direct resizing syntax but achieves similar *> results using 'reference modification' syntax: declare names2 as string occurs 7 set names2(0:size of names) to names *> Resizing to a smaller size is even simpler: set names2 to names[0:3] declare twoD as float-short occurs any, any. declare rows as binary-long = 3 declare cols as binary-long = 10 set size of twoD to rows, cols declare jagged = table of (table of binary-long(1 2) table of binary-long(3 4 5) table of binary-long(6 7 8 9)) *> Can also do: declare jagged2 as binary-long occurs any, occurs any set size of jagged2 to 3 set size of jagged2(1) to 5 set jagged2(1 5) to 5</pre>	<pre>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) ' Note, however, that this produces a new copy of the array -- ' it is not an in-place resize! ReDim Preserve names(6) Dim twoD(rows-1, cols-1) As Single twoD(2, 0) = 4.5 Dim jagged()() As Integer = { New Integer(2) {1, 2}, New Integer(3) {3, 4, 5}, New Integer(4) {6, 7, 8, 9}} jagged(0)(4) = 5</pre>	<pre>public class Arrays { public static void main(String args[]) { int nums[] = { 1, 2, 3 }; String names[] = new String[5]; names[0] = "David"; names[5] = "Bobby"; // throws ArrayIndexOutOfBoundsException // Can't resize arrays in Java String names2[]; // Copy elements from an array names2 = java.util.Arrays.copyOfRange(names, 0, 3); float twoD[][]; int rows = 3; int cols = 10; twoD = new float[rows][cols]; int[][] jagged = { { 1, 2 }, { 3, 4, 5 }, { 6, 7, 8, 9 } }; int[][] jagged2 = new int[3][]; jagged[0] = new int[5]; jagged[0][4] = 5; } }</pre>

Choices

C#	COBOL	VB.NET	Java
<pre>greeting = age < 20 ? "What's up?" : "Hello"; // Good practice is that all consequents are enclosed in {} // or are on the same line as if. 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. if (x > 5) { x == y; } else if (x == 5) { 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": r++; break; case "blue": b++; break; case "green": g++; break; default: other++; break; // break necessary on default }</pre>	<pre>declare age as binary-long = 10 declare greeting as string *>greeting = age < 20 ? has no directly equivalent syntax in COBOL if age < 20 move "What's up?" to greeting else move "Hello" to greeting end-if declare x as binary-long = 200 declare y as binary-long = 3 if x not = 100 and y < 5 multiply 5 by x multiply 2 by y end-if *> evaluate is preferred in COBOL rather than if/else if/else evaluate x when > 5 multiply y by x when 5 add y to x when < 10 subtract y from x when other divide y into x end-evaluate declare color as string = "blue" declare r b g other-color as binary-long evaluate color *> can be any type when "pink" add 1 to r when "blue" add 1 to b when "green" add 1 to g when other add 1 to other-color end-evaluate</pre>	<pre>greeting = IIf(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 And y < 5 Then x == 5 : y == 2 ' Preferred If x <> 100 And y < 5 Then x == 5 y == 2 End If ' To break up any long single line use _ If whenYouHaveAReally < longLine And _ itNeedsToBeBrokenInto2 > Lines Then _ UseTheUnderscore(charToBreakItUp) If x > 5 Then x == y ElseIf x = 5 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</pre>	<pre>public class choices { public static void main(String[] args) { int age = 10; String greeting = age < 20 ? "What's up?" : "Hello"; if (age < 20) { greeting = "What's up?"; } else { greeting = "Hello"; } int x = 200; int y = 3; if (x != 100 && y < 5) { x = 5 * x; y = 2 * y; } if (x > 5) { x = x * y; } else if (x == 5) { x = x + y; } else if (x < 10) { x = x - y; } else { x = x / y; } String color = "blue"; int r = 0, b = 0, g = 0, other_color = 0; if (color.equals("pink") color.equals("red")) { r++; } else if (color.equals("blue")) { b++; } else if (color.equals("green")) { other_color++; } } }</pre>

Classes Interfaces

C#	COBOL	VB.NET	Java
<pre>//Accessibility keywords public private internal protected protected internal static // Inheritance class FootballGame : Competition { ... } // Interface definition interface IAlarmClock { ... } // Extending an interface interface IAlarmClock : IClock { ... } // Interface implementation class WristWatch : IAlarmClock, ITimer { ... }</pre>	<pre>*> Accessibility keywords *public *private *internal *protected *protected internal *static class-id Competition. end class. *> Inheritance class-id FootballGame inherits type Competition. end class. *> Interface definition interface-id IClock. end interface. interface-id ITimer. end interface. *> Extending an interface interface-id IAlarmClock inherits type IClock. end interface. *> Interface implementation class-id WristWatch implements type IAlarmClock, type ITimer. end class.</pre>	<pre>' Accessibility keywords Public Private Friend Protected Protected Friend Shared ' Inheritance Class FootballGame Inherits Competition ... End Class ' Interface definition Interface IAlarmClock ... End Interface ' Extending an interface Interface IAlarmClock Inherits IClock ... End Interface ' Interface implementation Class WristWatch Implements IAlarmClock, ITimer ... End Class</pre>	<pre>//Accessibility keywords public private // The closest counterpart to .NET's "internal" is specified // by omitting the visibility keyword, though this "default" // visibility has some behaviour differences. protected static // Inheritance class FootballGame extends Competition { ... } // Interface definition interface IAlarmClock { ... } // Extending an interface interface IAlarmClock extends IClock { ... } // Interface implementation class WristWatch implements IAlarmClock, ITimer { ... }</pre>

Comments

C#	COBOL	VB.NET	Java
<pre>// Single line /* Multiple line */ /// <summary>XML comments on single line</summary> /** <summary>XML comments on multiple lines</summary> */</pre>	<pre>\$set sourceformat(variable) * Single line "old-style" COBOL comment with '*' in column 7 *> inline comment may follow COBOL statements etc. **> <summary>XML comments</summary></pre>	<pre>' Single line only REM Single line only ''' <summary>XML comments</summary></pre>	<pre>// Comments on a single line /* C style multi-line comments */</pre>

Constants

C#	COBOL	VB.NET	Java
<pre>const int MAX_STUDENTS = 25; // Can set to a const or var; may be initialized in a constructor readonly float MIN_DIAMETER = 4.93f;</pre>	<pre>class-id Students. * optionally public etc., and may reference constants in other classes 01 MAX_STUDENTS binary-long constant value 25 + 3. * Can set to a const or var; may be initialized in a constructor 01 MIN_DIAMETER float-short value 4.93 initialize only. method-id Main(). display MAX_STUDENTS display MIN_DIAMETER end method. end class.</pre>	<pre>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</pre>	<pre>public class constants { // constant value cannot be changed ; public static final int MAX_STUDENTS = 25 + 3; // This value can be assigned at declaration // OR // assigned in constructor // (not both) public final float MIN_DIAMETER = 4.93f; }</pre>

Constructors Destructors

C#	COBOL	VB.NET	Java
<pre>class SuperHero { private int _powerLevel; public SuperHero() { _powerLevel = 0; } public SuperHero(int powerLevel) { this._powerLevel = powerLevel; } ~SuperHero() { // Destructor code to free unmanaged resources. // Implicitly creates a Finalize method } }</pre>	<pre>class-id Heater. 01 _powerLevel binary-long. method-id new. set _powerLevel to 0 end method. method-id new (powerLevel as binary-long). set _powerLevel to powerLevel end method. \$if JVMGEN set method-id Finalize override protected. * JVM finalizer method. end method. \$else method-id Finalize override protected. * .NET finalizer method. end method. \$end end class.</pre>	<pre>Class SuperHero Private _powerLevel As Integer Public Sub New() _powerLevel = 0 End Sub Public Sub New(ByVal powerLevel As Integer) Me._powerLevel = powerLevel End Sub Protected Overrides Sub Finalize() ' Destructor code to free unmanaged resources MyBase.Finalize() End Sub End Class</pre>	<pre>public class ConstructorsDestructors { private int _powerLevel; public ConstructorsDestructors(int powerLevel) { _powerLevel = powerLevel; } @Override public void finalize() { // finalizer code } }</pre>

Data Types

C#	COBOL	VB.NET	Java
<pre>Value Types bool byte, sbyte char short, ushort, int, uint, long, ulong float, double decimal DateTime (a framework type) Reference Types object string Initializing bool correct = true; // Can also infer variable type if the value has a well-defined type var incorrect = false; // automatically a bool byte b = 0x2A; // hex // no support for octal literals // no support for binary literals object person = null; string name = "Dwight"; char grade = 'B'; DateTime now = DateTime.Now; // No support for date/time literals decimal amount = 35.99m; float gpa = 2.9f; double pi = 3.14159265; long lTotal = 123456L; short sTotal = 123; ushort usTotal = 123; uint uiTotal = 123; ulong ulTotal = 123; Type Information int x; Console.WriteLine(x.GetType()); // Prints System.Int32 Console.WriteLine(typeof(int)); // Prints System.Int32 Console.WriteLine(x.GetType().Name); // prints Int32 Type Conversion float d = 3.5f; int i = (int)d; // set to 3 (truncates decimal)</pre>	<pre>*Value Types *condition-value *binary-char (unsigned) *character *binary-short, binary-long, binary-double (unsigned) *float-short, float-long *decimal *DateTime (a framework type) *Reference types *object *string *Initializing declare correct as condition-value = true * Can also infer variable type if the value has a well-defined type declare incorrect = false * automatically a condition-value declare b as byte = h"2a" * hex declare o as byte = o"52" * octal declare b2 as byte = b"101010" * binary declare person as object = null declare nam as string = "Dwight" declare grade as character = "B" declare now as type DateTime = type DateTime::Now * No support for date/time literals declare amount as decimal = 35.99 declare gpa as float-short = 2.9 declare pi as float-long = 3.14159265 declare lTotal as binary-double = 123456 declare sTotal as binary-short = 123 declare usTotal as binary-short unsigned = 123 declare uiTotal as binary-long = 123 declare ulTotal as binary-long unsigned = 123 *Type Information declare x as binary-long display x::GetType * Prints System.Int32 display type of binary-long * Prints System.Int32 display x::GetType::Name * Prints Int32 *Type Conversion declare f as float-short = 3.5 * automatic conversion declare i = f as binary-long * set to 3 (truncates decimal) end program. program-id Legacy. * COBOL types not supported directly by other languages. * Visual COBOL supports these types on all platforms. * Only a few examples here 01 displayNumber pic 9(9).99. 01 computeNumber pic 9(9)V99. 01 alphaNumeric pic a(23). 01 binaryStorage pic x(12). * Also groups and redefines - a few examples 01 arecord. 03 aSubRecord pic x(10). 03 aUnion pic 9(10) redefines aSubRecord. end program.</pre>	<pre>Value Types Boolean Byte, SByte Char Short, UShort, Integer, UInteger, Long, ULong Single, Double Decimal Date Reference Types Object String Initializing Dim correct As Boolean = True ' No mechanism for automatic typing Dim b As Byte = &H2A 'hex Dim o As Byte = &O52 'octal ' No support for binary literals Dim person As Object = Nothing Dim name As String = "Dwight" Dim grade As Char = "B" Dim now As Date = DateTime.Now Dim past As Date = #12/31/2007 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 Type Information Dim x As Integer Console.WriteLine(x.GetType()) ' Prints System.Int32 Console.WriteLine(GetType(Integer)) ' Prints System.Int32 Console.WriteLine(TypeName(x)) ' Prints Integer Type Conversion 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)</pre>	<pre>import java.math.BigDecimal; import java.time.LocalDateTime; public class DataTypes { // Java doesn't have value types, though it does have // "primitives" available for the common types, which // are similar in some respects // boolean // byte // char // short, int, long // float, double // BigDecimal (a framework type) @SuppressWarnings("unused") public static void main(String[] args) { boolean correct = true; // no mechanism for automatic typing byte b = 0x2a; // hex byte o = 052; // octal byte b2 = 0b101010; // binary (Java 7 onwards) Object person = null; String name = "Dwight"; char grade = 'B'; LocalDateTime now = LocalDateTime.now(); // No support for date/time literals BigDecimal amount = new BigDecimal(35.99); float gpa = 2.9f; double pi = 3.14159265; long lTotal = 123456L; short sTotal = 123; // Java has no unsigned integer types // Java reflection does not work on primitive types. Integer x = new Integer(0); System.out.println(x.getClass().getCanonicalName()); // Type conversion short f = (short) 3.5; // must use cast to convert int i = f; // set to 3 (truncation) } }</pre>

Delegates Events

C#	COBOL	VB.NET	Java
<pre>delegate void MessageHandler(string message); class DelegatesEvents { static event MessageHandler MessageArrived; static void Main() { // explicit delegate construction new MessageHandler(MyHandler); // implicit delegate construction MessageHandler handler = MyHandler; // subscribe to an event MessageArrived += MyHandler; // raise the event MessageArrived("Test message"); // unsubscribe from the event MessageArrived -= MyHandler; // Throws a null reference exception as there are no subscribers MessageArrived("Test message 2"); // Safely raising an event MessageHandler handler2 = MessageArrived; if (handler2 != null) { handler2("Safe message"); } } static void MyHandler(string message) { System.Console.WriteLine(message); } }</pre>	<pre>delegate-id MessageHandler (str as string). end delegate. class-id DelegatesEvents. 01 MessageArrived event type MessageHandler static. method-id Main static. * explicit delegate construction invoke new MessageHandler(MyHandler) * implicit delegate construction declare handler as type MessageHandler = method MyHandler * subscribe to an event attach method MyHandler to MessageArrived * raise the event invoke MessageArrived("Test message") * unsubscribe from the event detach method MyHandler from MessageArrived * Throw a null reference exception as there are no subscribers invoke MessageArrived("Test message 2") * Safely raising an event declare handler2 as type MessageHandler = MessageArrived if handler2 not equals null invoke handler1("Safe message") end-if end method. method-id MyHandler static (str as string). display str end method. end class.</pre>	<pre>Delegate Sub MessageHandler(ByVal message As String) Class DelegatesEvents Shared Event MessageArrived As MessageHandler Shared Sub Main() ' explicit delegate construction new MessageHandler(AddressOf MyHandler) ' implicit delegate construction Dim messageHandler As MessageHandler = AddressOf MyHandler ' subscribe to an event AddHandler MessageArrived, AddressOf MyHandler ' fire the event RaiseEvent MessageArrived("Test message") ' unsubscribe from the event RemoveHandler MessageArrived, AddressOf MyHandler ' Doesn't throw an exception, even when there are no subscribers RaiseEvent MessageArrived("Test message 2") End Sub Shared Sub MyHandler(message As String) System.Console.WriteLine(message) End Sub End Class</pre>	<pre>// Java has no concept of delegates or events. // Similar behaviour is typically achieved by using // anonymous inner classes instead.</pre>

Enumeration

C#	COBOL	VB.NET	Java
<pre>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); // Prints "Stop is 1" Console.WriteLine((int) Status.Pass); // Prints 70 Console.WriteLine(Status.Pass); // Prints Pass</pre>	<pre>enum-id Action. 78 #Start. *> Start is a reserved word so use '#' symbol 78 #Stop. 78 #Rewind. 78 #Forward. end enum. enum-id Status. 78 Flunk value 50. 78 Pass value 70. 78 Excel value 90. end enum. class-id MainClass. method-id main static. declare a = type Action::Stop if a not = type Action::Start display a & " is " & a as binary-long *> Prints "Stop is 1" end-if display type Status::Pass as binary-long *> prints 70 display type Status::Pass *> prints Pass end method. end class.</pre>	<pre>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) ' Prints "Stop is 1" Console.WriteLine(Status.Pass) ' Prints 70 Console.WriteLine(Status.Pass.ToString()) ' Prints Pass</pre>	<pre>public class enumeration { public enum Action { Start, Stop, Rewind, Forward; } public enum Status { Flunk(50), Pass(70), Excel(90); } int mark; public int getMark() { return mark; } Status(int mark) { this.mark = mark; } public static void main(String[] args) { Action a = Action.Stop; if (a != Action.Start) { // Ordinal is printing out the 0-based index of // the entry rather than its "value" String index = new Integer(a.ordinal()).toString() System.out.println(a + " is " + index); } // can only print value if enum defines method System.out.println(Status.Pass.getMark()); System.out.println(Status.Pass); // prints Pass } }</pre>

Exception Handling

C#	COBOL	VB.NET	Java
<pre>// Throw an exception Exception ex = new Exception("Something is really wrong."); throw ex; // 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(); }</pre>	<pre>*> Throw an exception declare exc = new Exception("Something is really wrong."); raise exc *> Catch an exception declare x y as binary-long try declare o as string = null display o[0] *> display first character catch ex as type Exception display ex finally display "Finally" end-try</pre>	<pre>' Throw an exception Dim ex As 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)</pre>	<pre>public class exception_handling { public static void main(String[] args) throws Exception { Exception e = new Exception("Something is really wrong."); throw e; } public void nullReference() { try { String o = null; System.out.println(o.charAt(0)); } catch (Exception f) { System.out.println(f); } finally { System.out.println("Finally"); } } }</pre>

Functions

C#	COBOL	VB.NET	Java
<pre>// 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; } 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# supports optional arguments/parameters. // The default value is specified with the syntax '= value'. // Optional parameters must be listed last. void SayHello(string name, string prefix = "") { Console.WriteLine("Greetings, " + prefix + " " + name); }</pre>	<pre>class-id Functions. *> Pass by value (in, default), reference (in/out), and reference (out) method-id TestFunc (x as binary-long, reference y as binary-long, output z as binary-long). add 1 to x, y move 5 to z end method. method-id InstMethod. declare a as binary-long = 1 declare b as binary-long = 1 declare c as binary-long invoke TestFunc(value a reference b output c) *> Or invoke self::TestFunc(a b c) display a space b space c declare total as binary-long *> Commas in parameter lists are optional set total to function sum(4 3 2 1) *> returns 10 display total end method. method-id main static. declare i as binary-long set i to MySum(1 2 3 4) display i declare o = new self *> Parentheses are optional when there are no parameters. invoke o::InstMethod invoke SayHello("Strangelove", "Dr.") invoke SayHello("Madonna") end method. *> To create a non intrinsic variable argument list function: method-id MySum static (params nums as binary-long occurs any) returning mysum as binary-long. perform varying i as binary-long through nums add i to mysum end-perform end method. *> COBOL supports optional arguments/parameters. *> The default value is specified with the syntax '= value'. *> Optional parameters must be listed last. method-id SayHello static (nm as string, prefix as string = ""). display "Greetings, " prefix space nm end method. end class.</pre>	<pre>' 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 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 ' VB.NET supports optional arguments/parameters. ' The default value is specified with the syntax '= value', ' though optional parameters must be explicitly marked as such. ' Optional parameters must be listed last. Sub SayHello(ByVal name As String, Optional ByVal prefix As String = "") Console.WriteLine("Greetings, " & prefix & " " & name) End Sub SayHello("Strangelove", "Dr.") SayHello("Madonna")</pre>	<pre>public class functions { // Java only allows arguments to be passed by value public static void main(String[] args) { int i = mySum(1, 2, 3, 4); System.out.println(i); } // Use ellipses to indicate a variable argument list public static int mySum(int... arguments) { int sum = 0; for (int i : arguments) { sum += i; } return sum; } }</pre>

Loops

C#	COBOL	VB.NET	Java
<pre> 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 < 5; i++) { if (i < 4) { continue; } Console.WriteLine(i); // Only prints 4 } </pre>	<pre> declare c as binary-long = 0 perform 10 times display "Again and " end-perform *Pre-test loops: perform until c >= 10 add 1 to c end-perform perform varying c from 2 by 2 until c > 10 display c end-perform perform varying c2 as binary-long from 2 by 2 until c2 > 10 display c2 end-perform *Post-test loops: set c = 0 perform with test after until c >= 10 add 1 to c end-perform *> Varying *Array or collection looping declare names as string occurs any set content of names to ("Fred" "Sue" "Barney") perform varying s as string through names display s end-perform *Breaking out of loops: declare i as binary-long = 0 perform until exit if i = 5 exit perform end-if display i add 1 to i end-perform *Continue to next iteration: set i = 0 perform varying i from 0 by 1 until i >= 5 if i < 4 exit perform cycle end-if display i *Only prints 4 end-perform </pre>	<pre> 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 End If i += 1 End While Continue to next iteration For i = 0 To 4 If i < 4 Then Continue For End If Console.WriteLine(i) ' Only prints 4 Next </pre>	<pre> public class loops { public static void main(String[] args) { int c = 0 ; // Java has no direct equivalent to perform n times // pre test loops // "while" means condition inverted from COBOL's "until" while (c < 10); { c++; } for (int c2 = 2 ; c2 > 10 ; c2 += 2) { System.out.println(c2); } // Post test loops c = 0 ; do { c++; } while (c < 10) ; // looping through arrays or lists String names[] = {"Fred", "Sue", "Barney"}; for (String s : names) { System.out.println(s); } // break out of loops int i = 0; while (true) { if (i == 5) break; System.out.println(i); i++; } // Continue to next iteration: for (i = 0; i < 5; i++) { if (i < 4) continue; System.out.println(i); } } } </pre>

Namespaces

C#	COBOL	VB.NET	Java
<pre> namespace Harding.Compsci.Graphics { ... } // or namespace Harding { namespace Compsci { namespace Graphics { ... } } } using Harding.Compsci.Graphics; </pre>	<pre> *> At the file level: \$set ilnamespace(MicroFocus.COBOL.Examples) *> The directive can also be set at project *> level to apply the namespace to all types in the project. *> Alternatively, at the class level: class-id MicroFocus.COBOL.Examples.MyClass. end class. *> namespace import at file level: \$set ilusing(MicroFocus.COBOL.Examples) *> The directive can also be set at project *> level to apply the import to the entire project. </pre>	<pre> Namespace Harding.Compsci.Graphics ... End Namespace ' or Namespace Harding Namespace Compsci Namespace Graphics ... End Namespace End Namespace Imports Harding.Compsci.Graphics </pre>	<pre> package MicroFocus.COBOL.Examples; public class namespaces { } </pre>

Operators

C#	COBOL	VB.NET	Java
<pre> 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 + </pre>	<pre> *> Comparison operators: * = < > <= >= <> display (1 = 1) *> true display (1 > 2) *> false display (1 < 2) *> true display (1 <= 1) *> true display (1 >= 2) *> false display (0 < 1) *> true *> Arithmetic operators, * * / + - ** display ((5 + 1) * (5 - 1)) *> result 24 display (2 ** 3) *> result 8 display (function mod (5 3)) *> result 2 *> Local variable declarations declare a, b, c as binary-long declare d = "I'm a string" *> string type inferred from value *> Assignment statements *> move, set, compute set a = 1 move 2 to b compute c = 3 * 6 set a = 3 * 6 *> Type operator declare mystr as string = "string" declare myobj as object = mystr display (myobj instance of string) *> true *> Bitwise *> b-and, b-or, b-xor, b-not, b-left, b-right display (4 b-and 3) display (4 b-or 3) display (b-not 3) display (4 b-xor 3) display (4 b-left 1) *> shift left one place display (4 b-right 2) *> shift right two places *> Logical *> and, or, not display (true and false) *> false display (true or false) *> true display (not true) *> false </pre>	<pre> 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 & </pre>	<pre> public class operators { public static void main(String[] args) { // Comparison operators // = < > <= >= != System.out.println(1 == 1); // true System.out.println(1 > 2); // false System.out.println(1 < 2); // true System.out.println(1 <= 1); // true System.out.println(1 >= 2); // false System.out.println(0 != 1); // true // Arithmetic operators // * / + - // No exponentiation operator - use static method Math.pow(); System.out.println((5 + 1) * (5 - 1)); // result 24 System.out.println(Math.pow(2, 3)); // result 8 System.out.println(5 % 3); // result 2 // Local variable declarations. int a, b, c; String d = "I'm a string" ; // no declaration type inference // Assignment statements // all have same format in Java a = 1 ; b = 2; c = 3 * 6; a = 3 * 6; // type operator String mystr = "string"; Object o = mystr; System.out.println(o instanceof String); // true //Bitwise operations // & ^ ~ << >> System.out.println (4 & 3); System.out.println (4 3); System.out.println (~3); System.out.println (4 ^ 3); System.out.println (4 << 1); // shift left one place System.out.println (4 >> 2); // shift right two places // Logical // && ! System.out.println (true && false); // false System.out.println (true false); // true System.out.println (!true); // false } } </pre>

Program Structure

C#	COBOL	VB.NET	Java
<pre>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 + "!"); } } }</pre>	<pre>class-id MicroFocus.Examples.HelloWorld. *> member variable 01 field1 binary-long. method-id Main static(args as string occurs any). declare nam as string = "Bob" *> See if an argument was passed from the command line if size of args > 0 set nam to args[0] *> [] means 0 based index end-if end method. end class.</pre>	<pre>Imports System Namespace Hello Class HelloWorld Overloads Shared Sub Main(ByVal args() As String) Dim name As String = "VB.NET" Dim name As String = "VB.NET" ' See if an argument was passed from the command line If args.Length = 1 Then name = args(0) End If Console.WriteLine("Hello, " & name & "!") End Sub End Class End Namespace End Class</pre>	<pre>public class program_structure { // Member variable private String field1 ; public static void main(String[] args) { String nam = "Bob"; // See if an argument was passed from the command line if (args.length > 0) nam = args[0]; } }</pre>

Properties

C#	COBOL	VB.NET	Java
<pre>private int _size; public int Size { get { return _size; } set { if (value < 0) { _size = 0; } else { _size = value; } } } foo.Size++;</pre>	<pre>class-id Things. 01 _size binary-long private. 01 ReadOnly binary-long property with no set value 3. 01 ReadWrite binary-long property. property-id Size binary-long. *> Use property-value inside properties to *> pass the value in or out getter. set property-value to _size setter. if property-value < 0 set _size to 0 else set _size to property-value end-if end property. method-id main static. declare foo = new Things() add 1 to foo:Size display foo:Size display foo:ReadOnly set foo:ReadWrite to 22 end method. end class.</pre>	<pre>Private _size As Integer Public Property Size() As Integer Get Return _size End Get Set (ByVal Value As Integer) If Value < 0 Then _size = 0 Else _size = Value End If End Set End Property foo.Size += 1</pre>	<pre>// Java's properties are entirely convention based, // rather than being implemented by the system. // As a result they exist only as getter and setter methods. class Properties { private int _size; // no equivalent for shorthand properties public int getSize() { return _size; } public void setSize(int newSize) { _size = newSize; } public void static main(String[] args) { Properties foo = new Properties() foo.setSize(foo.getSize() + 1); System.out.println(foo.getSize()); } }</pre>

Strings

C#	COBOL	VB.NET	Java
<pre>// Escape sequences // \r // carriage-return // \n // line-feed // \t // tab // \\ // backslash // \" // quote // String concatenation string co = "Micro Focus\t"; co += "Ltd"; // "Micro Focus<tab>Ltd" // Chars char letter = school[0]; // letter is H letter = Convert.ToChar(65); // letter is A letter = (char)65; // same thing char[] letters = "abc".ToCharArray(); // letters now holds new[] { 'a', 'b', 'c' }; // Verbatim string literal string msg = @"File is c:\temp\x.dat"; // same as string msg = "File is 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) // s is "son" // Replacement s = mascot.Replace("sons", "nomial") // s is "Binomial" // Split string names = "Frank,Becky,Ethan,Braden"; string[] parts = names.Split(','); // 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 var buffer = new System.Text.StringBuilder("two "); buffer.Append("three "); buffer.Insert(0, "one "); buffer.Replace("two", "TWO"); Console.WriteLine(buffer); // Prints "one TWO three"</pre>	<pre>*> Escape sequences *> COBOL string literals don't have escape sequences, *> however you can specify strings as hex literals: *> x"0d" => carriage-return *> x"0a" => line-feed *> x"09" => tab *> \"V => backslash *> "" => quote *> String concatenation declare co as string = "Micro Focus" & x"09". set co to co & "Ltd" => "Micro Focus<tab>Ltd" *> Chars declare letter as character = co[0] *> letter is M set letter to 65 as character => "A" declare letters = as character occurs any = "abc".ToCharArray() => letters now holds table of character ('a', 'b', 'c') *> COBOL does not have verbatim string literals declare msg as string = "File is c:\temp\x.dat" *> String comparison declare town = "Newbury" *> this compares the value of *> strings rather than object references if town = "Newbury" display "true" end-if *> Substring display town[1:3] => displays "ewb" *> Replacement set s to mascot:replace("sons" "nomial") => s is "Binomial" display s *> Split declare names = "Frank,Becky,Stephen,Helen". declare parts as list[string] create p as binary-long create parts *> TODO: Provide JVM/.NET CONDITIONALLY COMPILED EXAMPLE OF SPLIT/REPLACE declare aParts as string occurs any declare nextPart as string perform until exit unstring names delimited by "," into nextPart with pointer p not on overflow exit perform write parts from nextPart end-unstring end-perform</pre>	<pre>' Special character constants (also accessible from ControlChars class) ' vbCr ' vbLf ' vbTab ' vbBack ' "" ' vbFormFeed ' vbVerticalTab ' vbCrLf, vbNewLine ' vbNullString ' String concatenation (use & or +) Dim co As String = "Micro Focus" & vbTab co = co & "Ltd" ' "Micro Focus<tab>Ltd" ' Chars Dim letter As Char = co.Chars(0) ' letter is H letter = Convert.ToChar(65) ' letter is A letter = Chr(65) ' same thing Dim letters = "abc".ToCharArray() ' letters now holds New Char(3) {"a", "b", "c"} ' No verbatim string literals Dim msg As String = "File is 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) ' s is "son" ' Replacement s = mascot.Replace("sons", "nomial") ' s is "Binomial" ' Split Dim names As String = "Frank,Becky,Ethan,Braden" Dim parts() As String = names.Split(",") ' One name in each slot ' Date to string Dim dt As New DateTime(1973, 10, 12) Dim s As String = 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"</pre>	<pre>public class Strings { public static void main(String[] args) { // Escape sequences: // \n // line-feed // \t // tab // \\ // backslash // \" // quote // string concatenation String co = "Micro Focus\t"; co = co + "Ltd"; // "Micro Focus<tab>Ltd" char letter = co.charAt(0); letter = (char) 65; // "A" // String literal String msg = "File is c:\\temp\\x.dat"; // String comparison String town = "Newbury"; if (town.equals("Newbury")) { System.out.println("true"); } // Substring System.out.println(town.substring(1, 3)); //Displays "ewb" // Split String names = "Frank,Becky,Stephen,Helen"; String[] parts = names.split(","); // split argument is a regex } }</pre>

Structs

C#	COBOL	VB.NET	Java
<pre>struct StudentRecord { public string name; public float gpa; public StudentRecord(string name, float gpa) { this.name = name; this.gpa = gpa; } } StudentRecord stu = new StudentRecord("Bob", 3.5f); StudentRecord stu2 = stu; stu2.name = "Sue"; Console.WriteLine(stu.name); // Prints Bob Console.WriteLine(stu2.name); // Prints Sue</pre>	<pre>valuetype-id StudentRecord. 01 #name string public. 01 averageMark float-short public. method-id new(studentName as string, averageMark as float-short). set self::name to studentName set self::averageMark to averageMark end method. end valuetype. class-id RecordClient. method-id main static. declare stu = new StudentRecord("Bob", 3.5). declare stu2 as type StudentRecord set stu2 to stu set stu2::name to "Sue" display stu::name => Prints Bob display stu2::name => Prints Sue end method. end class.</pre>	<pre>Structure StudentRecord 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 StudentRecord = New StudentRecord("Bob", 3.5) Dim stu2 As StudentRecord = stu stu2.name = "Sue" Console.WriteLine(stu.name) ' Prints Bob Console.WriteLine(stu2.name) ' Prints Sue</pre>	<pre>// Java has no equivalent to .NET's general purpose structs.</pre>

Using Objects

C#	COBOL	VB.NET	Java
<pre> SuperHero hero = new SuperHero(); // No "With" construct hero.Name = "SpamMan"; hero.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 ((line = reader.ReadLine()) != null) { Console.WriteLine(line); } } </pre>	<pre> declare hero = new SuperHero declare hero2 as type SuperHero declare obj as object declare lin as string *No "With" construct set hero::Name to "SpamMan" set hero::PowerLevel to 3 invoke hero::Defend("Laura Jones") invoke type SuperHero::Rest *> Calling static method set hero2 to hero *> Both reference the same object set hero2::Name to "WormWoman" display hero::Name *> Prints WormWoman set hero to null *> Free the object if hero = null set hero to new SuperHero end-if set obj to new SuperHero if obj is instance of type SuperHero display "Is a SuperHero object." end-if end program. class-id SuperHero. 01 #Name string property. 01 PowerLevel binary-long property. method-id Defend (defendee as string). end method. method-id Rest static. end method. end class. </pre>	<pre> 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() While Not line Is Nothing Console.WriteLine(line) line = reader.ReadLine() End While End Using </pre>	<pre> public class using_objects { public static void main(String[] args) { SuperHero hero = new SuperHero(); SuperHero hero2; Object obj; String lin; hero.setName("SpamMan"); hero.setPowerLevel(3); hero.defend("Laura Jones"); SuperHero.rest(); // Static method hero2 = hero; // Both reference same object hero2.setName("WormWoman"); System.out.println(hero.getName()); // prints "WormWoman" hero = null; if (hero == null) { hero = new SuperHero(); } obj = new SuperHero(); if (obj instanceof SuperHero) { System.out.println("Is a SuperHero object"); } } } class SuperHero { private String name; private int powerLevel ; public String getName() { return name; } public void setName(String name) { this.name = name; } public int getPowerLevel() { return powerLevel; } public void setPowerLevel(int powerLevel) { this.powerLevel = powerLevel; } public void defend(String defendee) { } public static void rest() { } } </pre>