Gliwice, March 2004

Visual Basic .NET

Short Overview

Marek Mittmann

Agenda

- Data types and operators
- Statements
- Arrays
- Classes and objects
- Properties and indexers
- Delegates and events

First program

```
' Hello world application
Module Hello
  Sub Main()
    MsgBox("Hello world!")
  End Sub
End Module
```

Console application

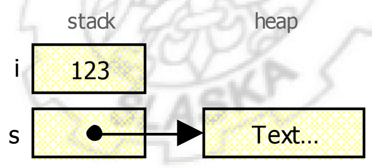
```
Imports System
Module Hello
  Sub Main(ByVal CmdArgs() As String)
    Dim I As Integer
    Console.WriteLine("Hello world!")
    For I = 0 To CmdArgs.GetUpperBound(0)
      Console.WriteLine("CmdArgs[{0}] = {1}", _
                        I, CmdArgs(I))
    Next I
  End Sub
End Module
```

VB.NET data types

Object	reference to object
String	sequence of Unicode characters
Char	Unicode character
Byte	8-bit integer value
Short	16-bit integer value
Integer	32-bit integer value
Long	64-bit integer value
Single	floating-point number (single precision)
Double	floating-point number (double precision)
Boolean	logical value (*True* or *False*)
Decimal	fixed-point decimal number
Date	date value

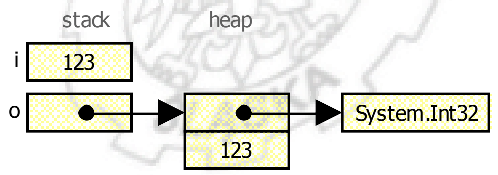
Value types and reference types

```
Dim i As Integer 'value type
Dim s As String  'reference type
i = 123
s = "Text..."
```



Boxing

```
Dim i As Integer = 123
Dim o As Object
o = i
```



Operators

Arithmetic	+ - * / \ Mod ^
Logical/bitwise	Not And Or Xor AndAlso OrElse
Concatenation	& +
Comparison	= <> < > <= >= Like Is
Assignment	= += -= *= /= \= ^= &=
Miscellaneous	AddressOf GetType

Namespaces

```
Namespace MyNamespace
  Namespace Inner
    Class MyClass
      Shared Sub Proc1()
       'procedure body
      En
    End
  End Na
End Name
```

```
Imports System, Microsoft.VisualBasic
Imports NamespaceAlias = MyNamespace.Inner

Module MyApp
  Sub Main()
    NamespaceAlias.Proc1()
  End Sub
End Module
```

Constants and variables

```
Sub Main()

  ' variables
  Dim r As Single = 1.25F
  Dim a, b As Integer
  Dim c As Integer = 2
  ' constants
  Const pi As Single = 3.14F

  a = 12 : b = 1
  Console.WriteLine(a + b + c)
  Console.WriteLine(pi * r * r)

End Sub
```

Enumerations

```
Enum Color
  Red          ' = 0
  Green = 10   ' = 10
  Blue         ' = 11
End Enum

Sub Main()

  Dim c As Color = Color.Green
  DrawBox2D(10, 20, c)
  DrawBox2D(12, 10, Color.Blue)

End Sub
```

Conditional statements

```
Sub Main(ByVal CmdArgs() As String)

  If CmdArgs.Length = 0
    Console.WriteLine("No arguments")
  ElseIf CmdArgs.Length = 1
    Console.WriteLine("Single argument")
  Else
    Console.WriteLine("{0} arguments", _
                      CmdArgs.Length)
  End If

End Sub
```

Select statement

```
Sub Main(ByVal CmdArgs() As String)

  Select Case CmdArgs.Length
    Case 1, 2
      Console.WriteLine("To few arguments")
    Case 3 To 5
      UseArgs1(CmdArgs)
    Case Is < 7
      UseArgs2(CmdArgs)
    Case Else
      Console.WriteLine("Error")
  End Select

End Sub
```

Loop statements

```
Dim I As Integer
For I = 0 To CmdArgs.Length-1
  Console.WriteLine(CmdArgs(I))
Next I

I = 0
While I < CmdArgs.Length
  Console.WriteLine(CmdArgs(I)) : I += 1
End While

I = 0
Do
  Console.WriteLine(CmdArgs(I)) : I += 1
Loop Until I >= CmdArgs.Length
```

Foreach statement

```
Sub Main()

  Dim Table() As Integer = New Integer() {2, 1, -5}
  Dim I As Integer

  For Each I In Table
    Console.WriteLine(I)
  Next I

End Sub
```

Exit statement

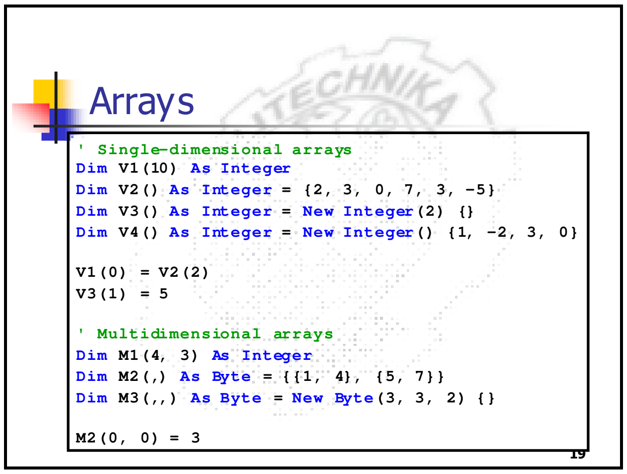
```
Sub Main(ByVal CmdArgs() As String)
    Dim I As Integer

    For I = 0 To CmdArgs.GetUpperBound(0)
      If CmdArgs(I) = "end" Then Exit For
      Console.WriteLine(CmdArgs(I))
    Next I

    I = 0
    While I < CmdArgs.Length
      If CmdArgs(I) = "end" Then Exit While
      Console.WriteLine(CmdArgs(I))
    End While
End Sub
```

Procedures and functions

```
' Function
Function Sum(ByVal A As Double, _
             ByVal B As Double) As Double
  Sum = a + b
End Function

' Procedure
Sub Main(ByVal CmdArgs() As String)
  If CmdArgs.Length < 2 Then Exit Sub

  Console.WriteLine(Sum(Val(CmdArgs(0)), _
                      Val(CmdArgs(1))))
End Sub
```

Exceptions

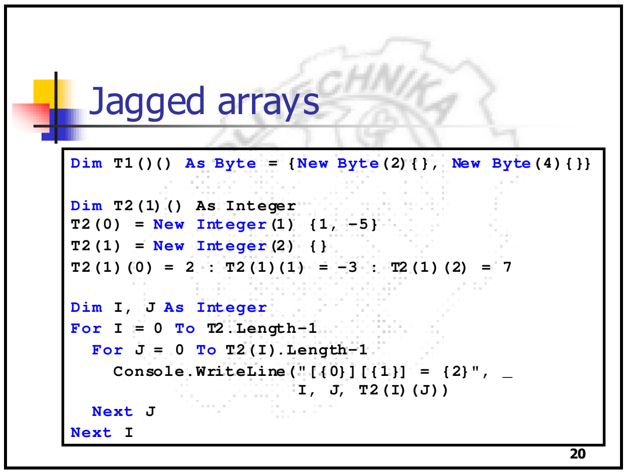
```
Function Div(ByVal A As Integer, _
             ByVal B As Integer) As Integer
  If B = 0 Then Throw New Exception("Divide by zero")
  Div = A / B
End Function

Sub Main(ByVal CmdArgs() As String)
  Try
    Console.WriteLine(Div(Val(CmdArgs(0)), _
                          Val(CmdArgs(1))))
  Catch Ex As Exception
    Console.WriteLine("Error: " & Ex.Message)
  Finally
    Console.Writeline("Done")
  End Try
End Sub
```



Arrays

```
' Single-dimensional arrays
Dim V1(10) As Integer
Dim V2() As Integer = {2, 3, 0, 7, 3, -5}
Dim V3() As Integer = New Integer(2) {}
Dim V4() As Integer = New Integer() {1, -2, 3, 0}

V1(0) = V2(2)
V3(1) = 5

' Multidimensional arrays
Dim M1(4, 3) As Integer
Dim M2(,) As Byte = {{1, 4}, {5, 7}}
Dim M3(,,) As Byte = New Byte(3, 3, 2) {}

M2(0, 0) = 3
```

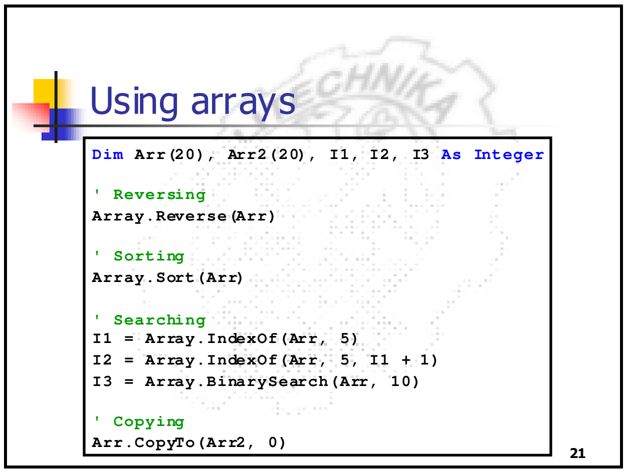
19



Jagged arrays

```
Dim T1()() As Byte = {New Byte(2){}, New Byte(4){}}

Dim T2(1)() As Integer
T2(0) = New Integer(1) {1, -5}
T2(1) = New Integer(2) {}
T2(1)(0) = 2 : T2(1)(1) = -3 : T2(1)(2) = 7

Dim I, J As Integer
For I = 0 To T2.Length-1
  For J = 0 To T2(I).Length-1
    Console.WriteLine("[{0}][{1}] = {2}", _
                      I, J, T2(I)(J))
  Next J
Next I
```

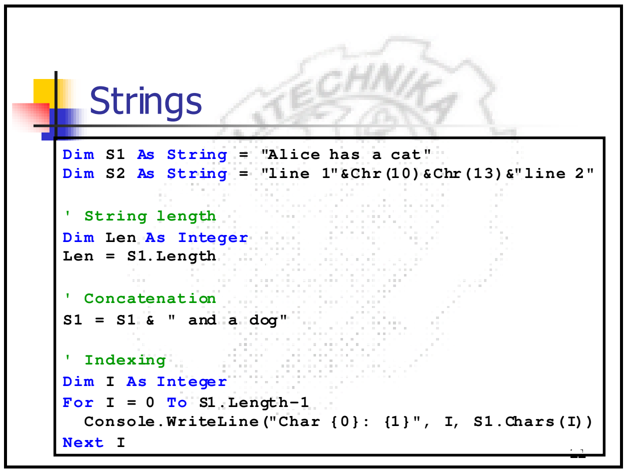
20



Using arrays

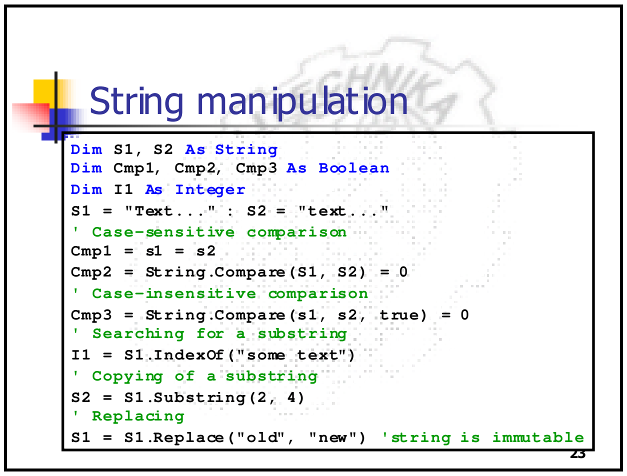
```
Dim Arr(20), Arr2(20), I1, I2, I3 As Integer

' Reversing
Array.Reverse(Arr)

' Sorting
Array.Sort(Arr)

' Searching
I1 = Array.IndexOf(Arr, 5)
I2 = Array.IndexOf(Arr, 5, I1 + 1)
I3 = Array.BinarySearch(Arr, 10)

' Copying
Arr.CopyTo(Arr2, 0)
```

21



Strings

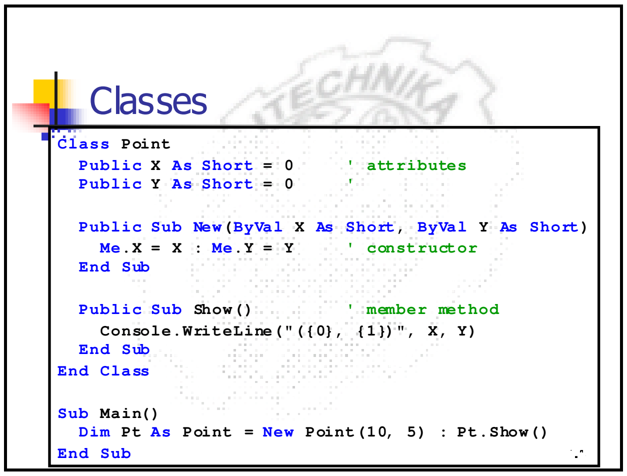
```
Dim S1 As String = "Alice has a cat"
Dim S2 As String = "line 1"&Chr(10)&Chr(13)&"line 2"

' String length
Dim Len As Integer
Len = S1.Length

' Concatenation
S1 = S1 & " and a dog"

' Indexing
Dim I As Integer
For I = 0 To S1.Length-1
  Console.WriteLine("Char {0}: {1}", I, S1.Chars(I))
Next I
```



String manipulation

```
Dim S1, S2 As String
Dim Cmp1, Cmp2, Cmp3 As Boolean
Dim I1 As Integer
S1 = "Text..." : S2 = "text..."
' Case-sensitive comparison
Cmp1 = s1 = s2
Cmp2 = String.Compare(S1, S2) = 0
' Case-insensitive comparison
Cmp3 = String.Compare(s1, s2, true) = 0
' Searching for a substring
I1 = S1.IndexOf("some text")
' Copying of a substring
S2 = S1.Substring(2, 4)
' Replacing
S1 = S1.Replace("old", "new") 'string is immutable
```

23



Classes

```
Class Point
  Public X As Short = 0      ' attributes
  Public Y As Short = 0      '

  Public Sub New(ByVal X As Short, ByVal Y As Short)
    Me.X = X : Me.Y = Y      ' constructor
  End Sub

  Public Sub Show()          ' member method
    Console.WriteLine("({0}, {1})", X, Y)
  End Sub
End Class

Sub Main()
  Dim Pt As Point = New Point(10, 5) : Pt.Show()
End Sub
```

With statement

```
Class Point3D
  Public X As Integer = 0
  Public Y As Integer = 0
  Public Z As Integer = 0
End Class

Sub Main()
  Dim Pt As Point3D = New Point3D()
  With Pt
    .X = 10
    .Y = -2
    .Z = 4
  End With
End Sub
```

25

Passing arguments

```
Class Point
  Dim X As Short = 0 : Dim Y As Short = 0
                    ' passing arguments by value
  Public Sub SetXY(ByVal X As Short, ByVal Y As Short)
    Me.X = X : Me.Y = Y
  End Sub
                    ' passing arguments by reference
  Public Sub GetXY(ByRef X As Short, ByRef Y As Short)
    X = Me.X : Y = Me.Y
  End Sub
End Class
' ...
Dim Pt As Point = New Point()
Dim X0, X1, Y1 As Short : X0 = 5 : X1 = 0 : Y1 = 0
Pt.SetXY(X0, 4)
Pt.GetXY(X1, Y1) ' after call X1 = 5 and Y1 = 4
```

Optional arguments

```
Class Computer
  Dim Id As String
  Dim Type As String

  Public Sub New(ByVal aId As String,
                 Optional ByVal aType As String = "PC")
     Id = aId : Type = aType
  End Sub
End Class

' ...

Dim C1 As Computer = New Computer("C001") ' Type = "PC"
Dim C2 As Computer = New Computer("C002", "Notebook")
```

27

Overloading

```
Class Point
  Public X As Short = 0
  Public Y As Short = 0

  Overloads Public Sub SetXY(ByVal X As Short, _
                             ByVal Y As Short)
    Me.X = X
    Me.Y = Y
  End Sub

  Overloads Public Sub SetXY(ByRef Pt As Point)
    X = Pt.X
    Y = Pt.Y
  End Sub
End Class
```

28

Inheritence

```
' Derived class
Class Point
  Inherits GraphObject
  Public X As Short = 0 : Public Y As Short = 0

  Public Sub New(ByVal Name As String, _
   ByVal X As Short, ByVal Y As Short)
     MyBase.New(Name)
     Me.X = X : Me.Y = Y
  End Sub

  Public Sub Show()
    Console.WriteLine("({0}, {1})", X, Y)
  End Sub
End Class
```

29

Virtual methods

```
Class Point
  Inherits GraphObject
  Public X As Short = 0
  Public Y As Short = 0

  Public Sub New(ByVal Name As String, _
   ByVal X As Short, ByVal Y As Short)
     MyBase.New(Name)
     Me.X = X : Me.Y = Y
  End Sub

  ' virtual method in derived class
  Public Overrides Sub Show()
    Console.WriteLine("{0}:({1}, {2})", Name, X, Y)
  End Sub
End Class
```

Abstract classes

```
' abstract class
MustInherit Class GraphObject
  Public Name As String

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

  ' pure-virtual method
  Public MustOverride Sub Show()
End Class
```

31

Interfaces

```
Interface IGraphObject
  Sub Show()
End Interface

Class Point
  Implements IGraphObject
  '...
  Public Sub Show() Implements IGraphObject.Show
    Console.WriteLine("({0}, {1})", X, Y)
  End Sub
End Class

Dim Pt As Point = New Point(2, 5)
Dim graphObj As IGraphObject = Pt
If Not graphObj Is Nothing Then graphObj.Show()
```

32

Members accessibility

- Accessibility modifiers for classes
  - **Friend** – accessible from the same module
  - **Public** – accessible from anywhere
- Accessibility modifiers for class members
  - **Public** – accessible from anywhere
  - **Protected** – accessible from the same class and from inherited classes
  - **Private** – only from within the same class
  - **Friend** – from the same module
  - **Protected Friend** – from the same module and from inherited classes

33

Constructor and destructor

```
Class ResourceWrapper
  Dim Handle As Integer = 0

  ' Constructor
  Public Sub New()
    Handle = GetWindowsResource()
  End Sub

  ' Destructor
  Protected Overrides Sub Finalize()
    ' Doesn't known, when it will be called
    FreeWindowsResource(Handle)
    MyBase.Finalize()
  End Sub
End Class
```

34

Interface *IDisposable*

```
Class ResourceWrapper : Implements IDisposable
  ' ...
  Private Sub DoDispose()
    FreeWindowsResource(Handle)
    Handle = 0
  End Sub

  Public Sub Dispose() Implements IDisposable.Dispose
    DoDispose()
    GC.SuppressFinalize(Me)
  End Sub

  Protected Overrides Sub Finalize()
    DoDispose()
  End Sub
End Class
```

Shared members

```
Class GraphObject
  Shared Counter As Integer = 0
  Public Name As String

  Public Sub New()
    Counter += 1
    Me.Name = "GraphObject" + Counter.ToString()
  End Sub

  Public Shared Sub ResetCounter()
    Counter = 0
  End Sub
End Class
```

36

Properties

```
Class Point
  Dim X As Short = 0
  Dim Y As Short = 0

  Public Property X() As Short
    Get
      Return X
    End Get
    Set(ByVal Value As Short)
      X = Value
    End Set
  End Property
End Class
```

Default properties

```
Class Worksheet
  Dim Data(20, 20) As Double

  Default Public Property Value(ByVal Col As String, _
   ByVal Row As Integer) As Double
     Get
       Return Data(Row, ColToIndex(Col))
    End Get
     Set(ByVal Value As Double)
       Data(Row, ColToIndex(Col)) = Value
     End Set
  End Property
End Class
' ...
Dim sheet As Worksheet = New Worksheet()
Sheet("A", 10) = 20.5
```

Delegates

```
Delegate Sub MyDelegate(ByVal Arg As String)

Class Tester
  Sub Proc(ByVal Arg As String)
    Console.WriteLine("Proc( {0} )", Arg)
  End Sub
End Class

Sub Main(ByVal CmdArgs() As String)
  Dim C As Tester = New Tester()
  Dim D As MyDelegate = AddressOf C.Proc

  D.Invoke(CmdArgs(0)) ' calls procedure
                       ' pointed by delegate
End Sub
```

Events

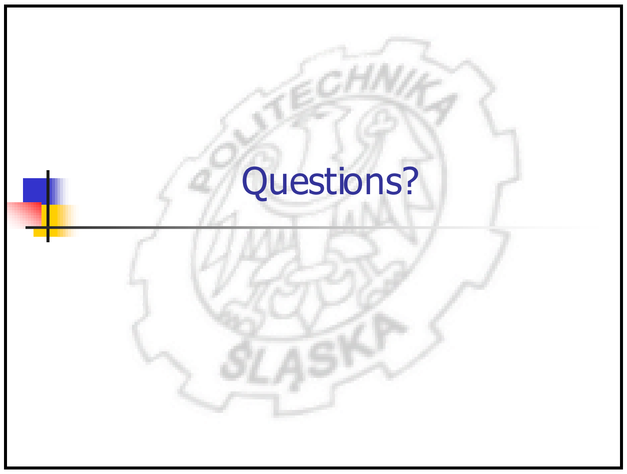
```
Class Button
  Public Event ClickEvent() ' points the event
  Public Sub PerformClick() ' raises the event
    RaiseEvent ClickEvent()
  End Sub
End Class

Public Sub OnClick()          ' event handler
  Console.WriteLine("Button clicked")
End Sub

Sub Main()
  Dim Bt As Button = new Button()
  AddHandler Bt.ClickEvent, AddressOf OnClick
  Bt.PerformClick()
End Sub
```

Event handlers

```
Friend WithEvents Button1 As Windows.Forms.Button

Protected Sub Button1_Click( _
  ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles Button1.Click

  MsgBox("Button clicked")
End Sub
```

Summary of classes, structures and interfaces

- Class
  - defines a set of properties, methods and events
  - reference type (allocated on the heap)
- Structure
  - like class can contains data and methods
  - value type (stored on the stack)
  - may not be inherited from
- Interface
  - similar to class, but do not provide implementation



Questions?