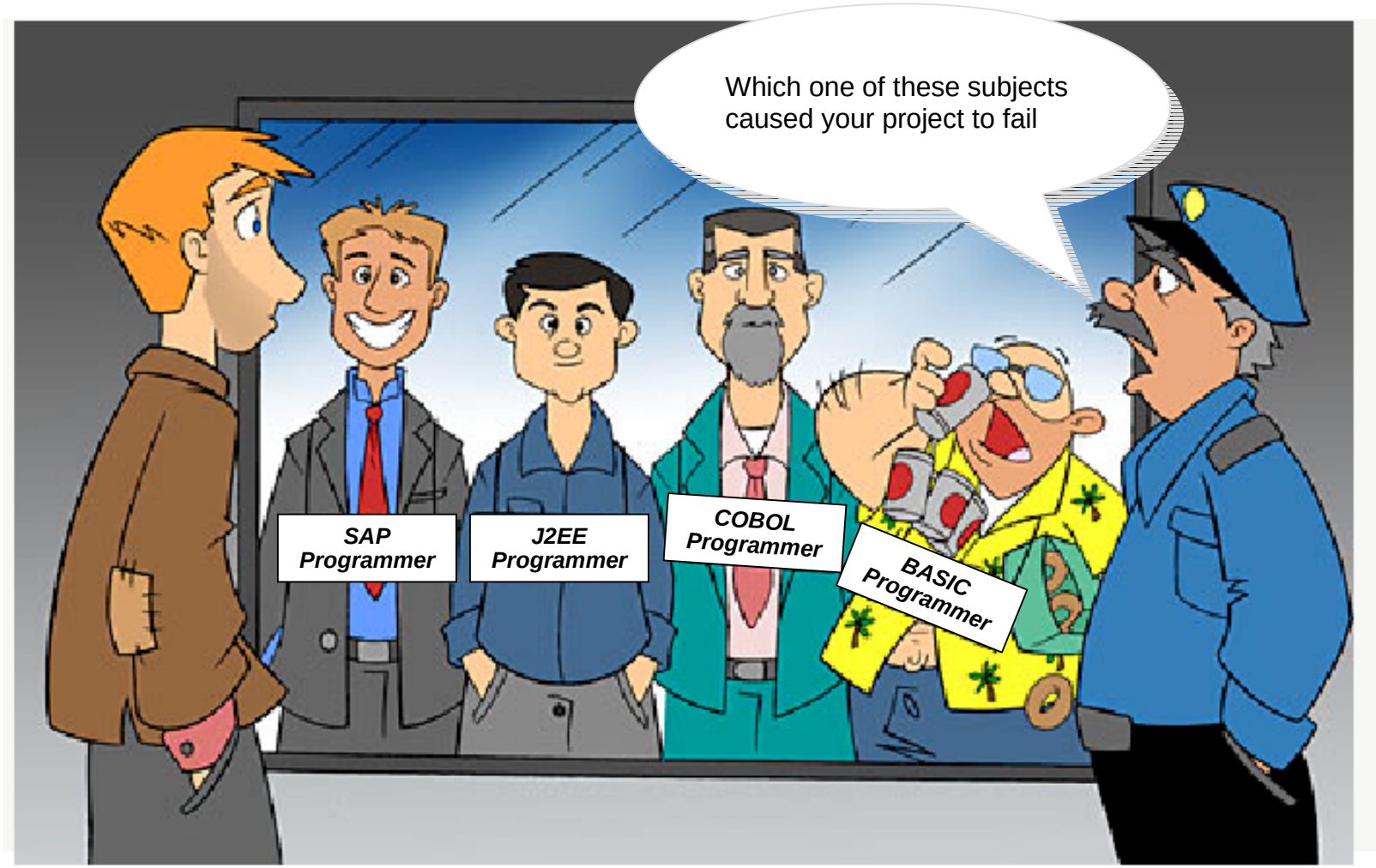
VB.NET mit Bezug auf VB6, VBA, VBS

School of Engineering

- Einführung
- Allgemeines, Datentypen
- Zuweisungen, Operatoren, Konvertierungen
- Kontrollstrukturen, Schleifen, GoTo
- Prozeduren, Parameterlisten
- Programmstruktur
- Klassen, Objekte, Konstruktoren, Destruktoren, Strukturen
- Felder, Eigenschaften, Methoden und Ereignisse
- Vererbung, Schnittstellen, Attribute

Frage


Which one of these subjects caused your project to fail
SAP Programmer
J2EE Programmer
COBOL Programmer
BASIC Programmer

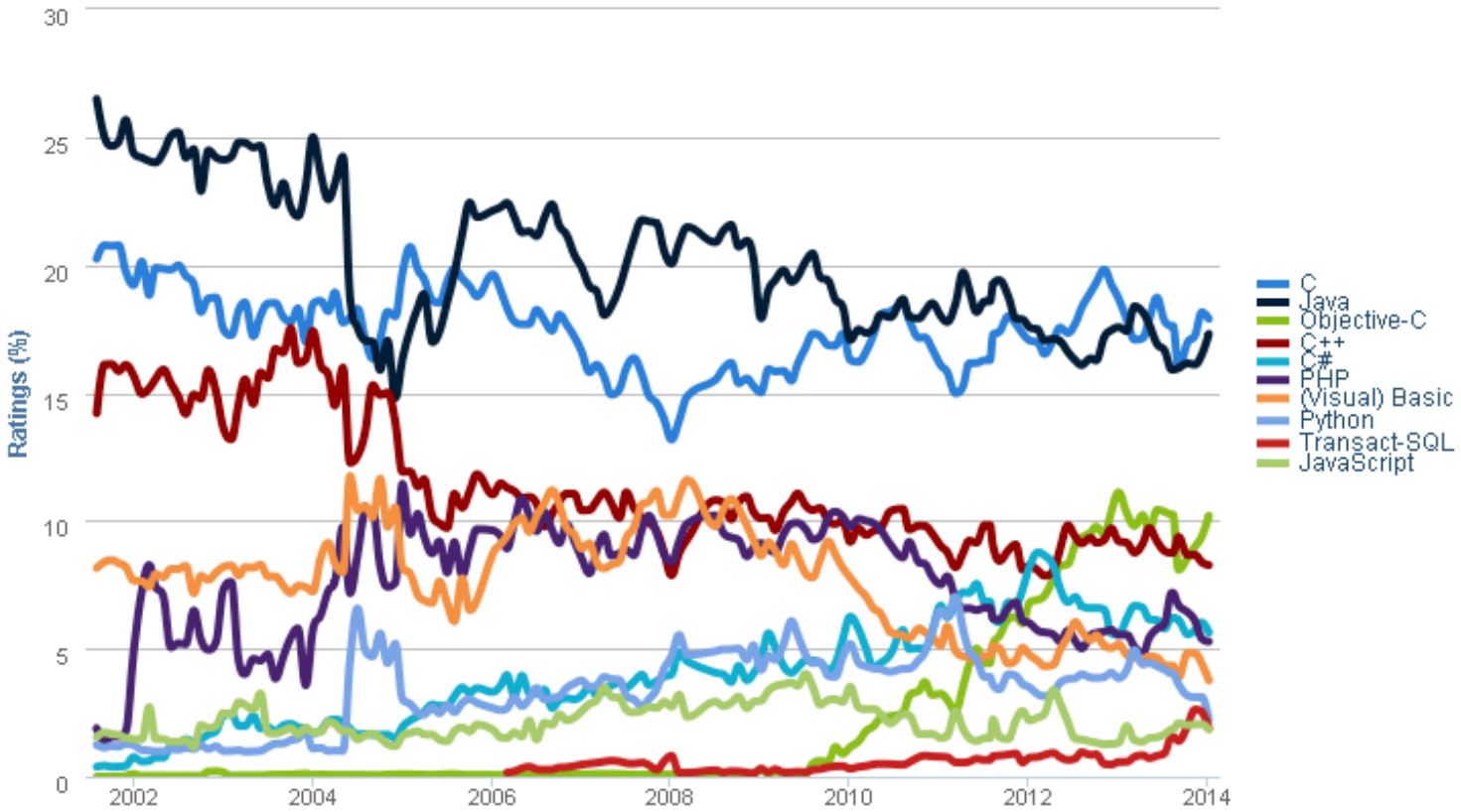
Geschichte von BASIC

- 1964 **BASIC**: **B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode
  - von Prof Kurtz und Kerneny als einfach zu erlernde Alternative für den Programmierunterricht entwickelt. Basic Programm wird interpretiert
  - Lediglich 14 Befehle, z.T. durch **FORTRAN** beeinflusst FOR,STEP
- 1975 Dennis Allison entwickelt Tiny-Basic für den Altair (erster PC)
- 1975 B. Gates & P. Allen entwickeln Altair BASIC -> Startpunkt von MS Erfolg
- 1980 Vielzahl von neuen BASIC Dialekten auf Microcomputer: Commodore C-64, IBM-kompatible PC's
- 1985ff Microsoft entwickelt **Q-** und **MBASIC** (später **GW BASIC**) für IBM-kompatible PC's
- 1991 Microsoft entwickelt **VisualBasic 1.0** für Windows System
- 1995 **VBS** als Alternative zu JavaScript in Explorer und CommandShell (heute noch)
- 1997 BASIC Varianten für diverse MS Office Produkte (Word Basic, Access Basic) werden zu VisualBasic for Application **VBA** vereinheitlicht
- 1998 **VisualBasic 6.0** für die BASIC Programmierung unter Windows; native-compilierter Code
- 2001 **VB.NET** als BASIC für .NET Umgebung

http://msdn.microsoft.com/downloads/
http://msdn.microsoft.com/officedev
http://msdn.microsoft.com/vba

Popularität von Programmiersprachen Dez. 13

- Gemessen anhand Aktivitäten auf dem Web und in Newsgroups: C# an 5. BASIC an 7.Stelle, Visual Basic.NET als Aufsteiger



http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Veänderungen seit letztem Jahr

School of Engineering

Dec 2013	Dec 2012	Change	Programming Language	Ratings	Change
1	1		C	17.890%	-0.81%
2	2		Java	17.311%	-0.26%
3	3		Objective-C	10.202%	-0.91%
4	4		C++	8.268%	-0.94%
5	5		C#	5.620%	+0.07%
6	6		PHP	5.281%	-0.26%
7	7		(Visual) Basic	3.752%	-1.42%
8	8		Python	2.210%	-1.64%
9	21	⇈	Transact-SQL	1.877%	+1.30%
10	11	↑	JavaScript	1.852%	+0.53%
11	15	⇈	Visual Basic .NET	1.688%	+0.80%
12	9	↓	Perl	1.072%	-1.10%
13	10	↓	Ruby	0.932%	-0.80%
14	17	↑	MATLAB	0.708%	+0.10%
15	12	↓	Delphi/Object Pascal	0.691%	-0.29%

Hello World

- Einfachstes VB.NET Programm

Als Klasse

```
Imports System
Namespace TestNamespace

  Public Class TestClass
   Public Shared Sub Main(ByVal args() as String)
     Console.WriteLine("Hello World!")
   End Sub
  End Class

End Namespace
```

Als Modul

```
Imports System
Module Main
  Sub Main()
    Console.WriteLine("Hello
World!")
  End Sub
End Module
```

mit vbc.exe übersetzen

```
> vbc.exe hello.vb
```

Allgemeines, Datentypen

Allgemeines

School of Engineering

- Es wird nicht zwischen Gross- und Kleinschreibung unterschieden, IDE "erzwingt" aber konsistente Schreibweise
- **Ende der Anweisung ist der Zeilenumbruch**; eine Anweisung über mehrere Zeilen wird mit _ am Ende der Zeile gekennzeichnet
- Zum **Trennen mehrerer Anweisungen** auf einer Zeile wird der : verwendet.
- Kommentare werden mit **'** oder **Rem** eingeleitet
- Weitere Regeln/Guidelines unter: http://www.gotdotnet.com/team/vb/VBSampleGuidelines.htm

Schlüsselwörter

- 136 Schlüsselwörter in VB.NET im Gegensatz zu ca. 80 Schlüsselwörtern in C#
- dürfen nicht für Variablen-Namen verwendet werden (mit [] erlaubt)

ADDHANDLER	ADDRESSOF	ANDALSO	ALIAS	AND		
ANSI	AS	ASSEMBLY	AUTO	BOOLEAN	BYREF	BYTE
	BYVAL	CALL	CASE	CATCH	CBOOL	CBYTE
	CCHAR	CDATE	CDEC	CDBLCHAR	CINT	CLASS
	CLNG	COBJ	CONST	CSHORT	CSNG	CSTR
	CTYPE	DATE	DECIMAL	DECLARE	DEFAULT	
DELEGATE	DIM	DIRECTCAST	DO	DOUBLE	EACH	ELSE
	ELSEIF	END	ENUM	ERASE	ERROR	EVENT
	EXIT	FALSE	FINALLY	FOR FRIEND	FUNCTION	GET
	GETTYPE	GOSUB	GOTO	HANDLES	IF	
IMPLEMENTS	IMPORTS	IN	INHERITS	INTEGER	INTERFACE	
IS	LET	LIB	LIKE	LONGLOOP	ME	MOD
	MODULE	MUSTINHERIT	MUSTOVERRIDE	MYBASE	MYCLASS	
NAMESPACE	NEW	NEXT				
NOT	NOTHING	NOTINHERITABLE	NOTOVERRIDABLE	OBJECT		
ON	OPTION	OPTIONAL	OR	ORELSE	OVERLOADS	
OVERRIDABLE	OVERRIDES	PARAMARRAY	PRESERVE	PRIVATE	PROPERTY	
PROTECTED	PUBLIC	RAISEEVENT	READONLY	REDIM	REM	
REMOVEHANDLER	RESUME	RETURN	SELECT	SET SHADOWS	SHARED	SHORT
	SINGLE	STATIC	STEP	STOPSTRING	STRUCTURE	SUB
	SYNCLOCK	THEN	THROW	TO TRUE	TRY	TYPEOF
	UNICODE	UNTIL	VARIANT	WHENWHILE	WITH	
WITHEVENTS	WRITEONLY					

Datentypen

- Elementare Datentypen
  - Ganzzahlen (Byte, Short, Integer, Long)
  - Gleitkommazahlen (Single, Double, Decimal)
  - alphanumerische Zeichen (Char, String)
  - sonstige (Boolean, Date)

keine vorzeichenlosen Typen

- **Value Types** VB.NET

```
Boolean
Byte
Char (example: "A"c)
Short, Integer, Long
Single, Double
Decimal
Date
```

- **Reference Types**

```
Object
String
```

- **Value Types** C#

```
bool
byte, sbyte
char (example: 'A')
short, ushort, int, uint, long, ulong
float, double
decimal
DateTime (not a built-in C# type)
```

- **Reference Types**

```
object
string
```

```
Dim x As Integer
Console.WriteLine(x.GetType()) 'Prints System.Int32
Console.WriteLine(TypeName(x)) 'Prints Integer
```

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

Enumerationen

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

Enum Action : Start : [Stop] :   End
Enum

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

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

Verwendung reservierter Wörter : []

```
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
```

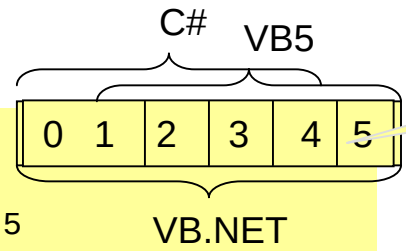
```
Action a = Action.Stop;

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

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

Variablen Deklaration

School of Engineering

- Deklaration von Variablen
  - Dim Variable1, Variable2 As *Datentyp*
  - Dim Variable3 As Integer = 1
- Deklaration von Arrays
  - Dim Array1() As String
  - Dim Array2 As Long()
- Initialisierung von Arrays
  - Dim Array3(10) As Long
  - Dim Kunden() As String = {“Maier“, “Hans“}
- Deklaration von mehrdimensionalen Arrays
  - Dim Array4(,,,) As String

Arrays



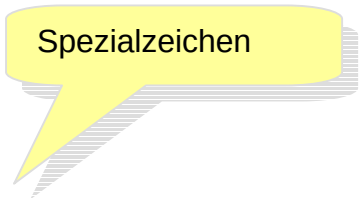
Ein Array wird ein Element grösser angelegt als in C#;
bis VB 5.0 begannen die Arrays bei 1 in VB.NET bei 0

```
Dim names(5) As String
names(0) = "David"
names(5) = "Bobby" ' 0..5


Dim nums() As Integer = {1, 2, 3}
' 4 is the index of the last element,
so it holds 5 elements
For i As Integer = 0 To nums.Length - 1
   Console.WriteLine(nums(i))
Next
' 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
```

```
string[] names = new string[5]; //0..4
names[0] = "David";
names[5] = "Bobby"; // Throws
System.IndexOutOfRangeException

int[] nums = {1, 2, 3};
// 5 is the size of the array
for (int i = 0; i < nums.Length; i++)
Console.WriteLine(nums[i]);


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

Konstanten Deklaration

- Elementare Datentypen
  - Ganzzahlen
    - 4, 4l, &H0004
  - Gleitkommazahlen
    - Single, Double, Decimal
    - Pi = 3.14
  - Referenzen:
    - Button: b = **Nothing**
  - alphanumerische Zeichen (Char, String)
    - String: "A", vbNullString
    - Char: "A"c, Chr(56), vbCr, vbCrLf, vbTab , vbBack, NullChar,..
  - Boolean
    - True, False




```
Const MAX_STUDENTS As Integer = 25
ReadOnly MIN_DIAMETER As Single = 4.93
```

```
const int MAX_STUDENTS = 25;
readonly float MIN_DIAMETER = 4.93f;
```

Zuweisungen, Operatoren, Konvertierungen

Zuweisung

- Über Zuweisungen erhalten Variablen einen Wert

```
Dim Variable1 As Double
Variable1 = 1.23456789
Dim Kunden(2) As String
Kunden(0) = “Maier“
Kunden(1) = “Hans“
Dim Variable2 As Klasse1
Variable2 = New Klasse1()

Set Variable2 = New Klasse1() ' VB6.0,VBA
```

in früheren BASIC Version mussten ref-Werte (Objekte) immer mit **Set** zugewiesen werden

Operatoren

- Arithmetische Operatoren (=, +, -, *, /, \, ^, Mod, &, +=, -=, *=, /=, \=, ^=, &=)
- Vergleichsoperatoren (<, >, =, < >, >=, <=, Like)
- Verkettungsoperatoren (&, +)
- Logische Operatoren (Not, And, Or, Xor, AndAlso, OrElse)
- Bitweise Operationen (Not, And, Or, Xor)

"VB .NET" Like "*.[M-O]??"

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

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

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

Bitwise
And Not Xor, Or << >>

Logical
AndAlso OrElse Not  And Or

Note: AndAlso and OrElse are for
short-circuiting 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
+
```

Typenumwandlung

In VB.NET immer

- Mit Option Strict On ähnlich zu C# - Off zu VBA,VBS,Python,Groovy
- implizite Konvertierung

  Dim i As Integer, d As Double

  i = 432

  d = i

  i = d 'erlaubt wenn Strict Off

Dynamic Typing

```
Dim i
i = 3
```

```
Dim i as Integer
i = 3
```

- explizite Konvertierung

  (CBool, CByte, CChar, CDate, CDbl, CDec, CInt, CLng, CObj, CShort, CSng, CStr)

  d = CDbl(i)

- allgemeine Umwandlungsfunktion CType

  d = CType(i, Double)

- für Buchstaben, Chr, Asc, ChrW, AscW

  c = Chr(56), i = Asc("a")

VB6.0 für Unicode Zeigen

```
Dim d As Single = 3.5
Dim i As Integer = CType(d, Integer)
i = CInt(d) ' same result as CType
i = Int(d) ' set to 3 (Int function
truncates the decimal)
```

```
float d = 3.5f;
int i = (int) d
```

Funktionsbibliotheken

Funktionsbibliotheken

- .NET Bibliotheken für
  - Array Umformungen: System.Array; Instanz-Methoden
  - Mathematische Funktionen: System.Math
  - String: System.String; Instanz-Methoden
  - Date: System.DateTime; Instanz-Methoden

- Zur einfacheren Portierung von bestehenden VB* Programmen stehen Bibliotheken zur Verfügung, die den alten Schnittstellen nachempfunden sind

- Mit ***Imports Microsoft.VisualBasic*** werden nachfolgende Funktionen zusätzlich verfügbar
- im Prinzip statische Methoden, aber
- speziell: **sie können ohne Angabe einer Basisklasse aufgerufen werden**
  - z.B. `UBound(a)` liefert den oberen Index eines Arrays

Array Funktionen

- **Erase** - Erases all values of an array
  - erase (arrayname)
- **Dim** - Creates an array
  - dim arrayname(25)
- **ReDim** - Resets the bounds of an array (has option to save values)
  - redim arrayname(28)
- **UBound** - Returns the upper dimension of an array
  - i = ubound (arrayname)
  - for i as Integer = LBound(a) to UBound(a)
- **LBound** - Returns the lower dimension of an array
  - i = lbound (arrayname)
- **Filter** - Returns a subset of an array based on a filter
  - Filter (inputarray, searchstring)
- **Array** -It returns an array that has been filled with data from a list.
  - ArrayName = Array (10, 20, 30)
- **IsArray** - return true if variable is an Array
- **Join** - Concatenates strings within an array

Mathematische Funktionen 1

- **Val** - Returns the numerical content of a string
  - `result = Val ("123.4")`
- **Int** - Returns an integer by truncating (for negative values return value that is less or equal)
  - `i = Int ( tempvariable )`
- **Fix** - Returns an integer by truncating (or negative values return value that is greater or equal)
  - `i = Fix ( tempvariable )`
- **Hex** - Returns the hexadecimal value of any number
  - `temp$ = Hex ( tempvariable )`
- **Oct** - Returns the octal value of any number
  - `temp$ = Oct ( tempvariable )`
- **Rnd** - Returns a random number between 0 and 1
  - `tempvariable1 = Rnd`
- **Randomize** - Initializes the Rnd function so it gives different answers each time
  - `Randomize`

Mathematische Funktionen 2

School of Engineering

- Seltener gebrauchte VB* Funtionen -> Math

- **Math.Sign** - Returns the sign of a number (früher **Sgn**)
  - `i = Math.Sign ( tempvariable )`
- **Math.Sin** - Returns the sine of an angle
  - `tempvariable1 = Math.Sin ( tempvariable2 )`
- **Math.Cos** - Returns the cosine of an angle
  - `tempvariable2 = Math.Cos ( tempvariable )`
- **Math.Abs** - Converts a number to a positive value
  - `i = Math.Abs ( tempvariable )`
- **Math.Sqrt** - Returns the square root of a number (früher **Sqr**)
  - `tempvariable1 = Math.Sqrt ( tempvariable2 )`
- **Math.Log** - Returns the base 10 logarithm of a number
  - `tempvariable1 = Math.Log ( tempvariable2 )`
- **Math.Atan** - Returns the arctangent of an angle (früher **Atn**)
  - `tempvariable1 = Math.Atan ( tempvariable )`
- **Math.Round** - Rounds a number to a selectable number of decimal places
  - `result = Math.Round ( tempvariable,2 )`
- **Math.Tan** - Returns the tangent of an angle
  - `tempvariable1 = Math.Tan ( tempvariable2 )`

String Funktionen 1

- **Left** - Returns the left n characters of a string
  - `temp$ = Left ( teststring$, 4 )`
- **Right** - Returns the right n characters of a string
  - `temp$ = Right ( teststring$, 4 )`
- **Trim** - Removes leading and trailing spaces of a string
  - `temp$ = Trim ( teststring$ )`
- **LTrim** - Removes only the leading spaces of a string
  - `temp$ = LTrim ( teststring$ )`
- **RTrim** - Removes only the trailing spaces of a string
  - `temp$ = RTrim ( teststring$ )`
- **UCase** - Makes all characters upper case
  - `temp$ = UCase ( teststring$ )`
- **LCase** - Makes all characters lower case
  - `temp$ = LCase ( teststring$ )`
- **Mid** - Returns n characters from a string, starting a any position
  - `temp$ = Mid ( teststring$, 1, 4 )`
- **Len** - Returns the length of a string (how many characters it has)
  - `temp$ = Len ( teststring$ )`
- **LSet** - Positions a string inside another, flush to the left
  - `temp$ = LRset ( teststring$ )`
- **RSet** - Positions a string inside another, flush to the right
  - `temp$ = RSet ( teststring$ )`
- **Format** - Returns a string formatted according to a user-defined format
  - `temp$ = Format ( teststring$, "####.0" )`

String Funktionen 2

- **Chr** - Returns the string representation of a number
  - `temp$ = Chr ( 32 )`
- **Asc** - Returns the ASCII code of a single character
  - `temp$ = Asc ( "A" )`
- **Space** - Returns n spaces
  - `temp$ = Space ( 15 )`
- **Instr** - Determines if one string is found within a second string
  - `i = Instr (starthere, string1, string2)`
- **InStrRev** - Determine if one string is found in a second, starting at the end
  - `i = InStrRev (string1, string2, start)`
- **StrComp** - Compares two strings
  - `result = StrComp (string1, string2)`
- **StrConv** - Converts the case of a string's characters
  - `StrConv (string, vbuppercase)`
- **StrReverse** - Reverses character order in a string
  - `StrReverse (string1)`
- **Replace** - Replaces each occurrence of a string
  - `Replace (bigstring, searchstring, replacementstring)`
- **FormatCurrency** - Returns a string using a currency format
  - `FormatCurrency(var1, 2)`
- **FormatDateTime** - Returns a date or time expression
  - `FormatDateTime("3/2/99",vbShortTime)`
- **FormatNumber** - Returns a number formatted according to a variety of options
  - `FormatNumber(var1, 2)`

Date Funktionen

- Date ist von VB.NET Sprache unterstützter Daten Typ -> System.DateTime

- **IsDate** test if variable is date Type
- **DateTime.Today**() Gets the current date (Date)
- **DateValue**(datetimeval) - Gets the date
- **TimeValue**(datetimeval)) - Gets the time
- **Now**() - Gets the current date and time (Now)
- **GetTimer**() - Returns the number of seconds since midnight (Timer)
- **DateAdd**(interval,num,dateval) - Adds a time interval to a date
- **DateDiff**(interval,date1,date2) - Returns how many time intervals there are between dates
- **DateSerial**(year,month,day) - Returns the month/day/year
- **DatePart**(interval,dateval) - Returns the date
- **Year**(dateval) - Returns the year
- **Month**(dateval) - Returns the month (integer 1..12)
- **MonthName**(dateval) - Returns the text of the name of a month
- **Day**(dateval) - Returns the day
- **Hour**(timeval) - Returns the hour
- **Minute** (timeval)- Returns the current minute
- **Second** (timeval)- Returns the current second
- **TimeSerial**(h,m,s) - Returns a date with the hour/minute/second
- **WeekDay** (datevalue)- Returns the current day of the week (integer 1..7)
- **WeekDayName**(datevalue) - Returns the text of a day of the week

Kontrollstrukturen, Schleifen, GoTo, Ausnahmen

Kontrollstrukturen

- Die If-Anweisung

```
If Bedingung1 Then
        ‘ Bedingung1 wahr
ElseIf Bedingung2 Then
        ‘ Bedingung1 falsch und Bedingung2 wahr
Else
        ‘ Bedingung1 falsch und Bedingung2 falsch
End If
```

```
' One line doesn't require "End If", no "Else"
If language = "VB.NET" Then langType = "verbose"

' 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
```

```
' or to break up any long single command 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
```

Kontrollstrukturen

- Die Select Case-Anweisung

```
Select Case Variable1
        Case Ausdruck1
                ‘ Variable1 hat den Wert Ausdruck1
        Case Ausdruck2
                ‘ Variable1 hat den Wert Ausdruck2
        Case Else
                ‘ Variable1 hat keinen der genannten Werte
End Select
```

```
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
```

Schleifen

- Die For...Next-Schleife

  For *Zählervariable* = *Anfangswert* To *Endwert* [Step *Inkrement*]

  ‘ Auszuführender Code

  Next *Zählervariable*

```
Dim names As String() = {"Fred", "Sue", "Barney"}
For i As Integer = LBound(names) To UBound(names)
  Console.WriteLine(s(i))
Next i
```

- Die For Each...Next-Schleife

  For Each *Elementvariable* In *Liste*

  ‘ Auszuführender Code

  Next *Elementvariable*

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

Schleifen

- Die While-Schleife

  While *boolescher Ausdruck*

  ‘ Auszuführender Code

  End While

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

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

- Die Do Until-Schleife

  Do Until *boolescher Ausdruck*

  ‘ Auszuführender Code

  Loop

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

- Die Do...Loop-Schleife 1

  Do

  ‘ Auszuführender Code

  Loop Until *boolescher Ausdruck*

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

- Die Do...Loop-Schleife 2

  Do

  ‘ Auszuführender Code

  Loop While *boolescher Ausdruck*

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

GoTo, Stop, Exit

School of Engineering

- Die Goto-Anweisung

  Goto *Label*

  ‘ Auszuführender Code

  Label : ‘ Auszuführender Code

Auf GoTo kann und sollte in VB.NET verzichtet werden

- Die Stop-Anweisung
  - Unterbruch des Programms
- Die Exit Do, For, Function, Property, Select, Sub, Try, While Anweisung
  - Verlässt bezeichneten Code-Block vorzeitig

```
While True
  c += 1
  If c = 10 Then Exit While
End While
```

Ausnahmebehandlung

- Unstrukturierte Ausnahmebehandlung (deprecated) UEH (Unstr. Ex. Handling)

  On Error GoTo ErrorHandler

  ErrorHandler: ‘ beim Eintreten einer Ausnahme auszuführender Code

```
On Error GoTo MyErrorHandler
...
MyErrorHandler:
Console.WriteLine(Err.Description)
```

- Strukturierte Ausnahmebehandlung (zwingend in Klassen) SEH (St. Ex. Handling)

```
Try
        ‘ Zu schützender Code
Catch Ausnahme As Ausnahmetyp
        ‘ beim Eintreten einer Ausnahme auszuführender Code
Finally
        ‘ dieser Code wird in allen Fällen ausgeführt,
End Try
```

- Auslösen mittels Throw

```
Throw New Exception ("Hallo");
```

Prozeduren, Parameterlisten

Prozeduren

- Visual Basic-Code wird immer in Prozeduren geschrieben
- Prozedurtypen in Visual Basic:
  - *Sub*-Prozeduren (Subroutine)
  - *Function*-Prozeduren
  - *Property*-Prozeduren
  - Ereignisbehandlungsprozeduren
- Prozeduren sind eingebunden entweder in
  - Module
  - Klassen

Sub-Prozedure, Subroutine

- Eine *Sub*-Prozedur (*Subroutine*) ist ein Block von Visual Basic .NET-Code, die zwischen dem Anweisungen Sub und End Sub eingeschlossen ist.

() sind optional

```
Sub Name()
        ‘ Code der Subroutine
[Exit Sub]

End Sub
```

```
Sub Test
lab1: x += 1
if  x = 10 then Exit
Sub
GoTo Lab1
End Sub

Test
```

```
Sub Test()
  x += 1
End Sub

Test()
```

Formale Parameter und Argumente

- Formale Parameterliste

```
Sub Name(Parameterliste)
        ‘ Code der Subroutine
End Sub
```

ByVal ist Default (in VB 6.0 / VBA ByRef!!!)

-
```
Sub Sub1(ByRef a As Long, ByVal b As Long, _
                          Optional c As Long = 0)
    a += b + c
End Sub
```

- Aufrufen

Call meist weggelassen

```
Dim x As Long = 0
Call Sub1(x, 1)
Sub1(x, 2, 1)
Call MsgBox(x)
```

```
' 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
```

ByRef muss nicht wiederholt werden

Optionale Parameter, variable Anzahl

- Optionale Parameter

  Gibt es nicht in C#

  Müssen am Schluss der Parameterliste sein und einen Default Wert besitzen

  Sub *Name*(*Optional [ByVal | ByRef] Parameter = Value*)

  ‘ Code der Subroutine

  End Sub

```
Sub SayHello(ByVal name As String, Optional ByVal prefix As
String = "")
Console.WriteLine("Greetings, " & prefix & " " & name)
End Sub
SayHello("Strangelove", "Dr.")
SayHello("Madonna")
```

Überladen

```
void SayHello(string name,
string prefix) {
Console.WriteLine("Greetings,
"
+ prefix + " " + name);
}

void SayHello(string name) {
SayHello(name, "");
}
```

- Variable Anzahl von Parametern

  Muss letzter Parameter sein

  Sub *Name*(*[ByVal | ByRef] ParamArray [ByVal | ByRef] Parameter*)

  ‘ Code der Subroutine

  End Sub

Function-Prozeduren, Rückgabewerte

- Eine *Function*-Prozedur ist ein Block von Visual Basic .NET-Code, die zwischen dem Anweisungen Function und End Function eingeschlossen ist, und einen Rückgabewert an das aufrufende Programm zurückgibt.

  Function *Name*[(*Parameterliste*)] As *Datentyp*
  Dim *Rückgabewert* As *Datentyp*
  ‘ sonstiger Code der Funktion
  Return *Rückgabewert*
  End Function

- Aufrufen

  Wert = Funktion1(100)

VB Style

```
Function Sum(ByVal ParamArray nums As Integer())
As Integer
  Sum = 0
  For Each i As Integer In nums
  Sum += i
  Next
End Functi
```

= Funktionsname

.NET Style

```
Function Sum(ByVal k As Integer) As
Integer
  return 15 +k
End Function
```

Programmstruktur

Programmstruktur

- Die Main-Prozedur
  - Jedes Visual Basic-Programm muss eine Prozedur mit der Bezeichnung Main enthalten
- Varianten von Main

```
Sub Main()
Sub Main(ByVal Args() As String)
Function Main() As Integer
Function Main(ByVal Args() As String) As Integer
```

- Hauptstruktur
  - Main (allg. Proceduren) muss in ein **Module** oder **Class** eingebettet sein
- Diese können in **Namespace** eingebettet sein

Module

- *Module* stellen eine einfache Möglichkeit zur Organisation von Hilfsfunktionen und globalen Daten bereit
- Module sind ein mit Klassen vergleichbarer Verweistyp
- Die Member eines Moduls sind implizit Shared (i.e. static)
- Module können nicht instanziert werden, können keine Schnittstellen implementieren, können nicht vererbt werden
- Deklaration

```
Module Modul1
        ‘ Code des Moduls
End Module
```

```
Imports System
Module MyModule
  Sub Main()
    Console.WriteLine("Hello World!")
  End Sub
End Module
```

Namensräume

- Ein Namensraum ist ein abstraktes Konzept, das verwendet wird, um eine Reihe von Klassen oder Modulen zusammenzufassen

```
Namespace Namensraum1
        Class Klasse1 ...
        Module Modul1 ...
End Namespace

Dim Variable1 As New Namensraum1.Klasse1
Variable1.Feld1 = Namensraum1.Modul1.Funktion1()
```

- Einbezug von fremden Namespace mit Imports

```
Imports System
Namespace TestNamespace

  public Class TestClass
   public shared Sub Main(ByVal args() as String)
     Console.WriteLine("Hello World!")
   End Sub
  End Class

End Namespace
```

Klassen, Objekte, Konstruktoren, Destruktoren, Strukturen

Klassen und Objekte

- Deklaration von Klassen

  [Public|Private|Friend] Class Klasse1

  ‘ restlicher Code der Klasse

  End Class

- Instanzieren

  Variable1 = new Klasse1

- Setzen zu Null

  Variable1 = Nothing

- Test auf Null

  If IsNothing(Variable1) Then Variable1 = new Klasse

  If Variable1 Is Nothing Then ....

- Typentest: TypeOf Obj Is Type

  If TypeOf Variable1 Is Klasse1 Then ...

- Zugriff auf eigene Instanz (this)
  - **Me**

```
Imports System.Collections
Dim Variable1 As Klasse1
Dim Variable2 As Hastable =  New Hashtable
Dim Variable2 As New Hashtable
Variable2 = Nothing
```

abgekürzte Schreibweise, bevorzugt

in VBS VarType Funktion

Konstruktoren

- Konstruktoren

```
Class Klasse1
        Public Sub New()
                ‘ Code zur Initialisierung
        End Sub

        Public Sub New(Variable1 As Datentyp)
                ‘ Code zur Initialisierung
        End Sub
End Class
```

```
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
End Class
```

Destruktoren

- Destruktoren

```
Class Klasse1
        ‘ wird vom Garbage Collector aufgerufen (NDD)
        ‘ NND = Nondeterministic Distruction
        Protected Overrides Sub Finalize()
                ‘ Code zum Aufräumen
        End Sub
End Class
```

```
Class SuperHero
Private _powerLevel As Integer
...

Protected Overrides Sub Finalize()
  ' Desctructor code to free unmanaged resources
   MyBase.Finalize()
End Sub

End Class
```

Strukturen

- *Strukturen* stellen ähnlich wie Klassen Datenstrukturen dar
- Verallgemeinerung des benutzerdefinierten Typs, der in älteren Versionen von Visual Basic unterstützt wird
- Strukturen sind im Gegensatz zu Klassen Wertetypen
- Strukturen können nicht vererbt werden
- Deklaration

```
Structure  Struktur1
        Public Variable1 As String
        Private Array1() As Long
End Stucture
```

```
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"
```

Zugriffsmodifikatoren

gelten für entsprechend für Klassen, Proceduren, Properties.

Schlüsselwort	Zugriff möglich:
Public	in der gesamten Anwendung
Protected	in der eigenen oder in einer abgeleiteten Klasse
Friend	in derselben Assembly, i.e. internal (C# internal)
Protected Friend	in der eigenen oder in einer abgeleiteten Klasse und in derselben Assembly
Private	innerhalb des Moduls oder der Klasse
Shared	auf Klassenebene (C# static)

Felder, Eigenschaften, Methoden und Ereignisse

Felder

- Deklaration von Feldern

```
Class Klasse1
        [Public|Private|Friend|Protected|Shared] Feld1 As Datentyp
End Class
```

- Zugriff auf Felder

```
        Dim Variable1 As New Klasse1
        Variable1.Feld1 = 0
```

Eigenschaften (Properties)

- Deklaration von Eigenschaften

```
Class Klasse1
    Private Variable1 As Datentyp
    Public Property Eigenschaft1() As Datentyp
        Get
            Return Variable1
        End Get
        Set (ByVal Value As Datentyp)
            Variable1 = Value
        End Set
End Class
```

```
Class Bar

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

End Class

foo.Size += 1
```

Methoden

- Deklaration von Methoden

```
Class Klasse1
        [Public|Private|Friend|Protected|Shared] Sub Methode1()
        End Sub
End Class
```

- Zugriff auf Methoden

```
Dim Variable1 As New Klasse1
Call Variable1. Methode1()
Dim Heute As Date
Heute = DateTime.Today()
```

Delegates

School of Engineering

- Typensichere Funktionszeiger
- Deklaration mit Delegate (ähnlich C#)
- Beim Aufruf die Invoke Methode der Delegate Variable verwendet
- Bei der Zuweisung AddressOf verwenden

```
Delegate Function CmpFunc(x As Integer, y As Integer) As Boolean

Public Function Cmp(x As Integer, y As Integer) As Boolean
... (This function implemented in some class)
End Function

Sub Sort(Sort As CmpFunc, ByRef IntArray() As Integer)
  ...
  If Sort.Invoke(IntArray(i), Value) Then
        ... Exchange values
  End If
  ...
End Sub

Call Sort( new CmpFunc( AddressOf aObj.Cmp), AnArray)
```

Ereignisse, Deklaration

- Ereignisse ermöglichen es einem Objekt, anderen Elementen mitzuteilen, dass etwas geschehen ist (z.B. Benutzeraktionen)
- Ein Ereignishandler ist eine *Sub*-Prozedur, die aufgerufen wird, wenn ein entsprechendes Ereignis auftritt

- Deklaration von Ereignissen über Delegate

```
Class Klasse1
        Public Delegate Sub Deleg1(Parameter)
        Public Event As Deleg1
End Class
```

- oder häufiger direkt (implizite Definition von Delegate)

```
Class Klasse1
        Public Event  Event1(Parameter)
End Class
```

Setzen und Aufruf von Ereignishandler

- Setzen von Ereignishandler: AddHandler, AdressOf

  AddHandler Variable1.Event1, AdressOf Variable1_Event1

- Aufruf von Ereignissen: RaiseEvent

```
Class Klasse1
        Public Event Event1(Parameter)
        Private Sub RaiseEvent1()
                ‘ Ereignis wird ausgelöst
                RaiseEvent Event1(...)
        End Sub
End Class
```

```
Event MsgArrivedEvent(ByVal message As String)

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

```
AddHandler Me.button1.Click, AddressOf Me.Button1Click

...
Private Sub Button1Click(sender As System.Object, e As
System.EventArgs)
    TextBox1.Text = "Hallo"
End Sub
```

Setzen von Ereignishandler, 2. Variante

- Setzen von Ereignisshandler mit der **Handles**-Klausel

```
Dim WithEvents Variable1 As New Klasse1

Sub Variable1_Event1() Handles Variable1.Event1
        ‘ Code der Ereignisbehandlungsprozedur
End Sub
```

```
Private WithEvents button1 As System.Windows.Forms.Button
...
Private Sub Button1Click(sender As Object,e As EventArgs) Handles button1.Click
    TextBox1.Text = "Hallo"
End Sub
```

Vererbung, Schnittstellen, Attribute

Vererbung

- Vererbung von Funktionalität

```
Public Class Klasse1
        ‘ Code der Klasse1
        Public Sub Methode1()
                ‘ Code der Methode
        End Sub
End Class

Public Class Klasse2
        Inherits Klasse1
        ‘ Code der Klasse2
End Class
```

...Vererbung - Erweitern & Überschreiben

- Erweiterung von Funktionalität

```
Public Class Klasse11
        Inherits Klasse1

        Public Function Methode2() As String
        End Function
End Class
```

- Überschreiben von Funktionalität

```
Public Class Klasse1
        Public Overridable Sub Methode1()
        End Sub
End Class
Public Class Klasse11
        Inherits Klasse1
        Public Overrides Sub Methode1()
                ' neue Implementierung
        End Sub
End Class
```

...Vererbung Überschatten

- Überschatten von Funktionalität

```
Public Class Klasse1
        Public Sub Methode1()
        End Sub
End Class
Public Class Klasse11
        Inherits Klasse1
        Public Shadows Sub Methode1()
                ‘ neue Implementierung
        End Sub
End Class
```

...Vererbung Aufruf der Oberklassenmethoden

- Oberklassen-Aufruf: MyBase
  - Basis-Methodenaufruf

    ```
    MyBase.Foo(...)
    ```

  - Basis-Konstruktoraufruf

    ```
    Public Sub New()
       MyBase.New()
          ‘ Code des Konstruktors
    End Sub
    ```

Überschatten vs. Überschreiben

Überschatten	Überschreiben
Schutz von späteren Änderungen der Basisklasse, durch die ein Member eingeführt wird, der bereits in der abgeleiteten Klasse definiert wurde	Gewährleisten des Polymorphismus durch die Definition einer anderen Implementierung einer Prozedur oder Eigenschaft mit derselben Aufrufabfolge
Wenn weder Shadows noch Overrides angegeben wurden wird Shadows angenommen	Overridable ist in der Basisklasse erforderlich. Overrides ist in der abgeleiteten Klasse erforderlich
Jeder deklarierter Elementtyp kann überschattet werden	Nur Prozeduren (Function oder Sub) oder Properties können überschrieben werden

Modifizierer zur Vererbung

- Versiegelte Klassen

```
Public NonInheritable Class Klasse1
        Public Shared Sub Methode1()
                ‘ Code der Methode1
        End Sub
End Class
```

- Abstrakte Klassen

```
Public MustInherit Class Klasse1
        Public MustOverride Sub Methode1()
End Class
```

Schnittstellen

- Deklaration von Schnittstellen

```
Public Interface Schnittstelle1
        Public Sub Methode1()
End Interface

Public Class Klasse1
      Implements Schnittstelle1

      Public Sub Methode1() Implements Schnittstelle1.Methode1
      End Sub
End Class
```

Attribute

School of Engineering

- Zusätzliche deklarative Informationen zum Programm
- Können auch selber definiert werden

```
Public Class PersonFirstNameAttribute
    Inherits Attribute
End Class
```

- Können zur Laufzeit mittels Reflection ausgewertet werden
- Erweitert die Programmfunktionalität
  - Gibt dem Laufzeitsystem Hinweise
  - Verwendet als Metaelemente

```
<WebMethod()>
Public Function  Hello As String ...
Dim <PersonFirstName()> Vorname As String
Dim <PersonFirstName()> PrimeiroNome As String
```

Nicht behandelt aber ebenfalls unterstützt

School of Engineering

- Reflection
- Generics
- Nullable Types
- Indexer
- ….

- VB.NET ist von den Sprachkonstrukten her gleichwertig zu C#

- Die Syntax ist aber gewöhnungsbedürftig

Schlussfolgerungen

- VB.NET als zweite .NET Programmiersprache
- Gemeinsames
  - Typensystem
  - Vererbungskonzept
  - Klassenbibliothek
  - Laufzeitsystem
- VB.NET und C# Code kann gleichzeitig und "nahtlos" in einem Projekt verwendet werden
- Gründe für Verwendung von VB.NET
  - Bestehende Software
  - Entwickler Know-how

Conclusion .NET Technology

School of Engineering

- **.NET programming languages** allow for various programming styles
  - Language Syntax: C#, VB.NET, ...
  - Object Oriented Programming: Classes, Structs
  - Aspect Oriented Programming: Attributes, partial methods
  - Functional Programming: Delegates, Lambda Expressions
  - Static and Dynamic Typing: var, dynamic
- The **.NET Class Library** is huge
- The **Visual Studio** development environment is powerful but fills up to 3+ GB and is increasingly complex to handle
- Microsoft extends and changes the language, library and tools rapidly
  - may become frustrating
  - e.g. H.M. wont write another .NET Technology book because he simply has not the time to catch up with MS frequency of changes

Development with .NET Technology is fun but challenging

Fragen ?

Quellen

- Michael Kolberg: “VB .NET Programmierung“, Franzis‘ Verlag GmbH, 2003
- “Programming in the .NET Environment“
- Jeff Prosise: “Microsoft .NET Entwicklerbuch“, Microsoft Press, 2002
- VB.NET Beispiele: http://msdn.microsoft.com/vbasic/downloads/samples/default.aspx

Anhang: File I/O VB Style

File I/O

School of Engineering

- Fille I/O durch Klassen aus System.IO

```
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 = 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()
```

- File I/O durch spezifische VB.NET Befehle (alt)
  - Es muss zuerst ein FileHandle (eine Zahl) erzeugt werden
  - Alle Operationen verlangen einen gültigen FileHandle

File I/O

- Dir - Returns a filename that matches a pattern
  - `temp$ = Dir ("*.*")`
- CurDir - Returns the current directory
  - `temp$ = CurDir()`
- MkDir - Creates a directory
  - `mkdir ( "newdirectoryname" )`
- ChDir - Changes the current directory to a new location
  - `chdir ( "newdirectoryname" )`
- ChDrive - Changes the current drive
  - `ChDrive("A")`
- RmDir - Removes the indicated directory
  - `rmdir ( "directoryname" )`
- Freefile - Returns an unused file handle
  - `i = FreeFile()`
- FileOpen (fnumber,fname,mode,access,share,recLen) - Opens a file for access, locking it from other applications (open)
  - `FileOpen( #1, "c:\test.txt", …)`
- FileClose - Closes a file so that other applications may access it (Close)
  - `FileClose (#1 )`
- LOF - Returns the length of a file in bytes
  - `i = lof ( #1 )`
- EOF - Returns a boolean value to indicate if the end of a file has been reached
  - `statusvariable = eof ( #1 )`

File I/O

- Rename - Renames a file (Name As)
  - `Rename("filename1","filename2" )`
- Kill - Deletes a file
  - `kill "filename"`
- Fileattr - Returns attribute information about a file
  - `i = FileAttr ( tempvariable )`
- GetAttr - Returns attributes of a file or directory
  - `i = GetAttr("c:\windows\temp")`
- SetAttr - Sets the attributes of a file
  - `SetAttr pathname, vbHidden`
- Reset - Closes all disk files opened by the FileOpen statement
  - `Reset ()`
- FileDateTime - Returns data file was created or last edited
  - `FileDateTime ( filename )`
- FileLen - Returns length of file in bytes
  - `FileLen ( filename )`
- FileCopy - Copies a file to a new name
  - `FileCopy sourcefile, destinationfile`
- Lock - Controls access to a part or all of a file opened by OPEN
  - `Lock ( #1)`
- UnLock - Restores access to a part or all of a file opended by OPEN
  - `UnLock ( #1)`

Text File I/O Funktionen

- Input - Reads ASCII text
  - input (#1, tempvariable$)
- WriteLine - Puts data in a file, with line separators for the data
  - WriteLine (#1, tempvariable$)
- Write - Puts data in a file
  - Write (#1, tempvariable$)
- Spc - Used in a print statement to move a number of spaces
  - Print (#2, var1; spc(15); var2)
- Tab - Used in a print statement to move to TAB locations
  - Write (#2, var1; Tab(20); var2)

File I/O Funktionen für binäre Dateien

- FileGet - Reads data from a file (Get)
  - get (#1, anyvariable)
- FilePut - Puts data into a file (Put)
  - put (#1, anyvariable)
- Seek - Moves the current pointer to a defined location in a file
  - seek (#1, 26)
- Input
  - input (#1, anyvariable)
- Loc - Returns current position with an open file
  - i = Loc(#2)

Anhang VBS VarType

The VarType function can return one of the following values:

```
0 = vbEmpty - Indicates Empty (uninitialized)
1 = vbNull - Indicates Null (no valid data)
2 = vbInteger - Indicates an integer
3 = vbLong - Indicates a long integer
4 = vbSingle - Indicates a single-precision floating-point number
5 = vbDouble - Indicates a double-precision floating-point number
6 = vbCurrency - Indicates a currency
7 = vbDate    - Indicates a date
8 = vbString - Indicates a string
9 = vbObject - Indicates an automation object
10 = vbError - Indicates an error
11 = vbBoolean - Indicates a boolean
12 = vbVariant - Indicates a variant (used only with arrays of Variants)
13 = vbDataObject - Indicates a data-access object
17 = vbByte - Indicates a byte
8192 = vbArray - Indicates an array
```

http://www.w3schools.com/vbscript/vbscript_ref_functions.asp