Visual Basic.NET

Programmierung

Zeitplan

Beginn	9.00 Uhr
Frühstück	10.30 Uhr - 10.50 Uhr
Mittag	12.30 Uhr - 13.30 Uhr
Kaffee	14.45 Uhr - 14.55 Uhr
Ende	16.00/30 Uhr

VB Programmierung - Inhalte

- Einführung in die Anwendungsentwicklung mit Visual Basic
- Entwicklungsumgebung Visual Studio
- Variablen, Datentypen und Operatoren
- Prozeduren und Programmablaufsteuerung
- Strukturierte Fehlerbehandlung und Debuggen
- Konzepte der Objektorientierten Programmierung
- Klassen, Objekte und Methoden
- Events und Eventhandler
- Strukturen und Schnittstellen
- Erstellen einfacher Windows Forms-Anwendungen
- Einsatz grundlegender Windows Forms-Steuerelemente

Ziele

- Möglichkeiten der Programmiersprache BASIC kennenlernen
- Strukturiertes-, prozedurales Programmieren
  - Zerlegung des Problemraums
  - Kontrollstrukturen: Sequenzen, Verzweigungen, Schleifen
- Konzepte der objektorientierten Programmierung verstehen
- Mit Objekten arbeiten und Zugriffsmethoden erstellen
- Bestehenden Programmcode lesen und analysieren können
- Grafische Schnittstelle programmieren

Links

- http://www.tiobe.com/tiobe_index
- http://openbook.rheinwerk-verlag.de/einstieg_vb_2012/
- https://msdn.microsoft.com/en-us/library/xk24xdbe(v=vs.90).aspx
- https://msdn.microsoft.com/en-us/library/sh9ywfdk.aspx

Visual Basic

Visual Basic is a third-generation event-driven programming language and integrated development environment from Microsoft for its COM programming model first released in 1991 and declared legacy in 2008.

Microsoft intended Visual Basic to be relatively easy to learn and use.

Visual Basic was derived from BASIC and enables the rapid application development (RAD) of graphical user interface (GUI) applications, access to databases using Data Access Objects, Remote Data Objects, or ActiveX Data Objects, and creation of ActiveX controls and objects.

A programmer can create an application using the components provided by the Visual Basic program itself.

Over time the community of programmers developed third party components.

Programs written in Visual Basic can also use the Windows API, which requires external function declarations.

The final release was version 6 in 1998 (now known simply as Visual Basic).

A dialect of Visual Basic, Visual Basic for Applications (VBA), is used as a macro or scripting language within several Microsoft applications, including Microsoft Office.

https://en.wikipedia.org/wiki/Visual_Basic

Build Process - Schema

```
source code: .vb
            ↓
            ↓ PREPROCESSOR
            ↓
          vb code
            ↓
libraries → ↓ COMPILER
            ↓
     CIL Bytecode
     • common intermediate language
     • assembly, executable code (debug, release)
     • contains metadata (5)
        • describe types, members, references
        • locate and load classes
        • lay out instances in memory
        • resolve method invocations
        • run-time context boundaries
            ↓
            ↓
 CLR(common language runtime) (9)
 • JIT (just-in-time compiler)
 • producing and executing native code
 • managed execution environment, managed code
 • load and run
 • enforce type safety
 • array bound and index checking
 • exception handling
 • garbage collection
 • threads
```

```
alternatively: NGEN(native image generator)
        → compiler outputs native code
        • faster program execution
        • needs still runtime environment
        • loss of portability
```

Invoke compiler from command line (vbc)

1. Start cmd.exe: win+r, cmd
2. Create project folder on desktop: cd Desktop, md folder, cd folder
3. edit kommandozeile.vb (shift u. 1 ≙ +)

```
Module Main
    Sub Main(args As String())
        System.Console.WriteLine("Hello " + args(0) + _
            ", today is " + System.DateTime.Now)
    End Sub
End Module
```

4. Compiler:
   C:\Windows\Microsoft.NET\Framework\v4.0.30319\vbc kommandozeile.vb
5. Aufruf: kommandozeile username

IDE

Intro - Microsoft Visual Studio (Administrator)

File Edit View Project Build Debug Team Tools Test Analyze Window Help

Solution Explorer

Solution 'Intro' (1 project)
Intro
My Project
Fakturierung.vb
Intro.vb

Projekt-Explorer

Source.cs | Intro.vb*

```
'==========================================================
' VISUAL BASIC
'==========================================================
'
'   Visual Basic is a third-generation event-driven programming la
'   integrated development environment from Microsoft for its COM
'   model first rele                                  cy in 2008.
'   Microsoft intend                                  ly easy to learn
'   Visual Basic was derived from BASIC and enables the rapid appli
'   development (RAD) of graphical user interface (GUI) applicatio
'   databases using Data Access Objects, Remote Data Objects, or A
'   Objects, and creation of ActiveX controls and objects.
'   A programmer can create an application using the components pr
'   Visual Basic program itself.
'   Over time the community of programmers developed third party c
'   Programs written in Visual Basic can also use the Windows API,
'   requires external function declarations.
'   The final release was version 6 in 1998 (now known simply as V
```

Programmbereich

107 %

Output

Show output from: Build

```
1>------ Build started: Project: Intro, Configuration: Debug Any CPU ------
1>C:\Users\test\Desktop\VB\Intro\Intro.vb(353,13): warning BC42024: Unused local variable: 'au8NumberLi
1>C:\Users\test\Desktop\VB\Intro\Int                 42024: Unused local variable: 'rgd'.
1>C:\Users\test\Desktop\VB\Intro\Int                 42024: Unused local variable: 'loop'.
1>C:\Users\test\Desktop\VB\Intro\Int                 42312: XML documentation comments must pre
1>  Intro -> C:\Users\test\Desktop\VB\Intro\bin\Debug\Intro.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Fehlerliste

Properties Error List Output

Class View

<Search>

Intro
Project References
Intro
Main

Klassenansicht

Ready Ln 11 Col 3 Ch 3 INS Publish

IDE - Tastenkombinationen

addnewitem	ctrl+shift+a
run	ctrl+f5
debug	f5
togbreakpoint	f9
stepinto	f11
stepover	f10
gotonexterrortag	f8
gotodefinition	f12
peekdefinition	alt+f12
completeword	ctrl+space
codesnippet	ctrl+k+x
moveline	alt+↑↓
insertlineabove	ctrl+enter
incsearch	ctrl+i
nextoccurence	f3
replacenext	alt+r
rename	ctrl+r+r
gotoblockbegend	ctrl+shift+↑↓
lowercase	ctrl+u
uppercase	ctrl+shift+u
comment	ctrl+k+c
uncomment	ctrl+k+u
worddeletetoend	ctrl+del
linecut	ctrl+l
properties	alt+enter, f4
showparameterinf	ctrl+shift+space
encapsulatefield	ctrl+r+e

nextwindow	ctrl+tab, ctrl+f6
nextsplitpane	f6
nextcategory	ctrl+pgdn
solutionexplorer	ctrl+alt+l
classview	ctrl+shift+c
collapseexpand	ctrl+m+m
collapseexpall	ctrl+m+l
togbookmark	ctrl+k+k
nextbookmark	ctrl+k+n
togwhitespace	ctrl+shift+8
formatdocument	ctrl+k+d

IDE - Autoergänzung

- Bsp.: Variable

```
Public Sub Print()
    Dim anzBestellungen = 0
    ...
    anz
End Sub
```

Tab-Taste ergänzt zu `anzBestellungen`

IDE - Einstellungen u. Snippets

Einstellungen

```
project/properties/build/warning level/level4
tools/options/text editor/basic/line numbers
tools/options/projects and solutions/always show error list...
tools/options/projects and solutions/show output...
tools/customize/commands/toolbar/debug/add command/debug/start without debugging
```

Code snippets(tools/code snippets manager, ctrl+k+x)

Anwendung: snippet + tab bzw. 2 x tab

applog	filreadbin	propwrite
appstop	filwritetext	secdecrypt
arrloc	for	secencrypt
arrsort	foreach	select
cbarraylist	fuction	struct
cbcomplete	funcgeneric	sub
cbdata	funcpararr	subover
coliter	generic	syspower
coliterdict	idispose	sysres
condown	ifelse	systime
conreadport	imagebyte	tbarray
doloopun	lbclear	testc
enum	lbdate	testm
enumcustom	mathrand	tryc
enumstr	opadd	trycf
event	osuser	typedate
except	pbiterate	typeremove
filcreatefile	propdef	typestrbyte
filcreatefold	property	typetime
filexistfile	propread	

IDE - Einstellungen u. Snippets

```
User defined snippet, xml-file, extension *.snippet
tools/code snippets manager/basic/import
<?xml version="1.0" encoding="UTF-8"?>
<CodeSnippets xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
  <CodeSnippet Format="1.0.0">
    <Header>
      <Title>WriteLine</Title>
      <Shortcut>cw</Shortcut>
    </Header>
    <Snippet>
      <Code Language="VB"><![CDATA[System.Console.WriteLine("$end$")]]></Code>
    </Snippet>
  </CodeSnippet>
</CodeSnippets>
```

Main-Methode (ohne Argumente)

```
Module Main
    Sub Main() ' startroutine of the module
        ' set startroutine
        ' 1. solution explorer
        ' 2. solution/project name/my project (wrench) 2xclick → properties
         (or project/properties)
        ' 3. startup object: module name

        ' output ide
        Global.System.Diagnostics.Debug.Print("debug output")
        ' (only in debug configuration, in release configuration, the debug
        ' elements will not be compiled into the code)

        ' output at command prompt (command line interpreter)
        Global.System.Console.WriteLine("console output")
        Global.System.Console.WriteLine("Hello {0}, today Is {1}",
                                        Global.System.Environment.UserName,
                                        Global.System.DateTime.Now)
        ' short form
        Console.WriteLine("Hello {0}, today Is {1}",
                          Environment.UserName,
                          Now)
    End Sub
End Module
```

Main-Methode (mit Argumente)

```
Module Main
    Sub Main(args() As String)
    ' setting:
    ' project/properties/debug/command line arguments
        For Each item As String In args
            System.Console.WriteLine(item)
        Next
    End Sub
End Module
```

Preprocessor Directives

```
#If False Then
Module M1
    Sub Main()
        System.Console.WriteLine("Today is " + WeekdayName( Weekday(Today) ) )
    End Sub
End Module
#End If

#If True Then
Imports Microsoft.VisualBasic.DateAndTime

Module M2
    Sub Main()
        Microsoft.VisualBasic.Interaction.MsgBox("Current Month: " + _
                                                 MonthName( Month(Today) ) )
    End Sub
End Module
#End If
```

Preprocessor Directives

```
#Const SELECTOR = False
Imports Microsoft.VisualBasic.DateAndTime

Module Main
    Sub Main()
        Dim culture As New System.Globalization.CultureInfo("en-US") ' de-DE, fr-FR
        Dim engFormat As System.Globalization.DateTimeFormatInfo = culture.DateTimeFormat
#If SELECTOR Then
        System.Console.WriteLine("Today Is " + engFormat.GetDayName(Weekday(Today)))
#Else
        System.Console.WriteLine("Current Month: " + engFormat.GetMonthName(Month(Today)))
#End If
    End Sub
End Module
```

Preprocessor Directives

```
#Const SELECTOR = 3
Imports Microsoft.VisualBasic.DateAndTime

Module Main
    Sub Main()
#If SELECTOR = 1 Then
        System.Console.WriteLine("Current Hour: " + Hour(Now).ToString)
#ElseIf SELECTOR = 2 Then
        System.Console.WriteLine("Current Minute: " + Minute(Now).ToString)
#Else
        System.Console.WriteLine("Current Second: " + Second(Now).ToString)
#End If
    End Sub
End Module
```

Identifiers

```
Naming conventions
• begin with alphabetic character or _
• must only contain alphabetic characters, decimal digits, and underscores
• must contain at least one alphabetic character or decimal digit if it
  begins with an underscore
• must not be more than 1023 characters long
• keywords can't be used as a identifiers (unless escaped)
'
Capitalization Styles (Naming Guidelines, MSDN)
+--------------------------+--------+----------------+
|        Identifier        |  Case  |    Example     |
+--------------------------+--------+----------------+
| Class                    | Pascal | AppDomain      |
| Enum type                | Pascal | ErrorLevel     |
| Enum values              | Pascal | FatalError     |
| Event                    | Pascal | ValueChange    |
| Exception class          | Pascal | WebException   |
| Read-only Static field   | Pascal | RedValue       |
| Interface                | Pascal | IDisposable    |
| Method                   | Pascal | ToString       |
| Namespace                | Pascal | System.Drawing |
| Parameter                | Camel  | typeName       |
| Property                 | Pascal | BackColor      |
| Protected instance field | Camel  | redValue       |
| Public instance field    | Pascal | RedValue       |
+--------------------------+--------+----------------+
```

Naming Conventions

```
Module NamingConventions
    Sub test()
        Dim UpperCamelCase = 0 ' or PascalCase
        Dim lowerCamelCase = 0

        ' keywords are normally not allowed for variable designation
        ' except usage of [ ]
        ' Dim select = 1 ' error
        ' Dim loop       ' error
        Dim [select] = 1 ' ok, escaped keyword
        Dim [loop] ' ok

        ' Hungarian notation, Systems Hungarian
        ' prefix encodes the actual data type
        Dim nOrderNum = 1000     ' integer
        Dim lAccountNum = 112233 ' long
        Dim bMarried = True ' boolean
        Dim yAge = 30       ' byte
        Dim chGroup = "C"   ' char
        Dim wItemId = 7788  ' word
        Dim au8NumberList() As Byte ' array of unsigned 8-bit integers

        ' Hungarian notation, Apps Hungarian
        ' encode the logical data type rather than the physical data type
        Dim rwPosition = 1  ' row
        Dim fPassed = False ' flag
        Dim rgd() As Double ' range, array with double values
        Dim cul = 100       ' count, number of elements, type unsigned long
    End Sub
End Module
```

Übung - Bezeichner

a) Welche der folgenden Variablendefinitionen sind korrekt?

```
___882
_888
_
item-nr
max
Max
/m/
abc$
gerundeterWert
durchschnittØ
Ødurchschnitt
AB123__45
12ABC
µm
ölstand
Frage?
hello☺
[Dim]
Dim
grad
grad°
bit
byte
bytebyte
Gauß
a!
text§1
Nr.
hundert€
hundertx10
zehnÅ
Bruch½
```

Lösung - Bezeichner

a)

```
Dim ___882 = 0          ' ok
Dim _888 = 0            ' ok
Dim _ = 0               ' ok
Dim item-nr = 0         ' error
Dim max = 0             ' ok
Dim Max = 0             ' error
Dim /m/= 0              ' error
Dim abc$ = 0            ' ok
Dim gerundeterWert = 0 ' ok
Dim durchschnittØ = 0  ' ok
Dim Ødurchschnitt = 0  ' ok
Dim AB123__45 = 0       ' ok
Dim 12ABC = 0           ' error
Dim µm = 0              ' ok
Dim ölstand = 0         ' ok
Dim Frage? = 0          ' ok
Dim hello☺ = 0          ' error
Dim [Dim] = 0           ' ok
Dim Dim = 0             ' error
Dim grad = 0            ' ok
Dim grad° = 0           ' error
Dim bit = 0             ' ok
Dim byte = 0            ' error
Dim bytebyte = 0        ' ok
Dim Gauß = 0            ' ok
Dim a! = 0              ' ok
Dim text§1 = 0          ' error
Dim Nr. = 0             ' error
Dim hundert€ = 100      ' error
Dim hundertx10 = 1000   ' ok
Dim zehnÅ = 10          ' ok
Dim Bruch½ = 0.5        ' error
```

Primitive data types, value types

- All numeric data types
- Boolean, Char and Date
- All structures, even if their members are reference types
- Enumerations, since their underlying type is always SByte, Short, Integer, Long, Byte, UShort, UInteger, or ULong

Visual Basic type	CLR type	Storage allocation
Boolean	Boolean	Depends on platform
Byte	Byte	1 byte
Char (single character)	Char	2 bytes
Date	DateTime	8 bytes
Decimal	Decimal	16 bytes
Double (double-precision)	Double	8 bytes
Integer	Int32	4 bytes
Long (long integer)	Int64	8 bytes
Object	Object (class)	4 bytes on 32-bit platform
SByte	SByte	1 byte
Short (short integer)	Int16	2 bytes
Single (single-precision)	Single	4 bytes
String (variable-length)	String (class)	Depends on platform
UInteger	UInt32	4 bytes
ULong	UInt64	8 bytes
User-Defined (structure)	(inherits ValueType)	Depends on platform
UShort	UInt16	2 bytes

Data Type Summary →https://msdn.microsoft.com/en-us/library/47zceaw7.aspx
→https://en.wikibooks.org/wiki/Visual_Basic/Data_Types

Value types, Integral types

```
Internal representation
-----------------------
Einerkomplement
1. Vorzeichenbit MSB
2. Restbits invertieren
Problem: Doppelte Darstellung der 0 (pos. und neg)

Zweierkomplement
1. Zahl vorzeichenlos
2. Bits invertieren
3. 1 addieren

Idee: Komplementärzahl zu einer Zahl, die addiert 0 ergibt,
• Prüfung des Vorzeichenbits entfällt
• keine doppelte Darstellung der 0
• keine zusätzliche Steuerlogik

z.B. (Wortlänge 3) :  010 + 101 = 111 + 1 = 0
=> Komplement zu 010(2) (= 2(10)) ist 110(2) (= -2(10))
Rechnen mit Restklassen Mod 256 und finden geeigneter Repräsentanten
(z.B. 0..255 äq -128..127 sind Rep. dieser 256 Restklassen bei Wortlänge 8)

Veranschaulichung: Alle Bits haben die gleiche Wertigkeit wie bei
positiver Darstellung. Nur das MSB erhält die negative Wertigkeit.

Variante (Verlassen des Wertebereichs, Addition der negativen Zahl)
Bsp.: Wortlänge 3 Bits, damit nächster Wert = 2³ = 8 = 1000(2),
Darstellung der Zahl -1(10) im 2er-Komplement => 8 + (-1) = 7(10) = 111(2)

Zuordnungen (Aufteilung des Wertebereichs In pos. und neg. Werte) :
(10) (1er-K) (2er-K)
 0    0 000   0 000
 1    1 001   1 001
 2    2 010   2 010
 3    3 011   3 011
 4   -1 110  -1 111 (=8-1)
 5   -2 101  -2 110 (=8-2)
 6   -3 100  -3 101 (=8-3)
 7    0 111  -4 100 (=8-4)
Wertebereich des 2er-K:  -4..+3, 4 negative, 4 positive Zahlen (incl. 0)
```

Value types, Integral types

```
Module BasicDatatypes
    Sub integralTypes()
        '----------------------------
        ' byte
        '----------------------------
        Dim by As Byte  ' zero-initialized
        Debug.Print(by) ' 0
        Debug.Print(Byte.MinValue) ' 0
        Debug.Print(Byte.MaxValue) ' 255
        by = 255
        Try
            by += 1 ' runtime error, OverflowException
        Catch e As System.Exception
            Debug.Print(e.ToString)
        End Try
        ' by = 256 ' buildtime error

        ' corresponding type in the .NET Framework
        Dim byStruct As System.Byte ' a structure
        byStruct = by
        Debug.Print(by.CompareTo(byStruct))    ' 0, equal
        Debug.Print(by.GetType.ToString)       ' System.Byte
        Debug.Print(byStruct.GetType.ToString) ' System.Byte

        Dim sby As SByte
        Dim sbyStruct As System.SByte
        Debug.Print(SByte.MinValue)        ' -128
        Debug.Print(System.SByte.MaxValue) '  127
        ' sby = SByte.MaxValue + 1 ' buildtime error
        sby = SByte.MaxValue
        Try
            sby += 1 ' runtime error
        Catch e As Exception
            Debug.Print(e.ToString)
        End Try

        ' sby = System.Convert.ToByte(128) ' runtime error
```

Value types, Integral types

```
'-----------------------------
' char
'-----------------------------
Dim c As Char = "ABC" ' ok but stores only A
c = "Z"  ' automatic conversion string to single unicode character
c = "Z"c ' specified as a char literal
c = "ABC"
' c = "ABC"c ' error, only one character
Debug.Print(c)      ' A
Debug.Print(Global.Microsoft.VisualBasic.Strings.Asc(c)) ' 65
c = Char.MinValue   ' 0
c = Char.MaxValue   ' &hffff (=65535)
Dim smiley = Global.Microsoft.VisualBasic.Strings.Chr(1)
System.Console.WriteLine("smiley=" + smiley)

' corresponding type in the .NET Framework
Dim cStruct As System.Char

'-----------------------------
' short
'-----------------------------
Dim s As Short
Debug.Print(Short.MinValue) ' -32768
Debug.Print(Short.MaxValue) '  32767
s = 22S ' literal type
Dim i16 As System.Int16 ' corresponding .NET type

Dim us As UShort
Debug.Print(UShort.MaxValue) ' 65535
us = 22US ' literal type
Dim uint16 As System.UInt16 ' corresponding .NET type
```

Value types, Integral types

```
'-----------------------------
' integer
'-----------------------------
Dim i As Integer
Debug.Print(Integer.MinValue) ' -2.147.483.648
Debug.Print(Integer.MaxValue) '  2.147.483.647
i = 1I  ' literal type
Dim j% ' shorthand
Dim i32 As System.Int32 ' corresponding .NET type

Dim ui As UInteger
Debug.Print(UInteger.MaxValue) ' 4.294.967.295
Dim ui32 As System.UInt32 ' corresponding .NET type
Debug.Print(System.UInt32.MaxValue)

Dim l As Long
Debug.Print(Long.MinValue) ' -9.223.372.036.854.775.808
Debug.Print(Long.MaxValue) '  9.223.37.203.685.477.5807
l = 22L
Dim ll& ' shorthand
Dim i64 As System.Int64 ' corresponding .NET type

Dim ul As ULong
Debug.Print(ULong.MaxValue) ' 18.446.744.073.709.551.615
Dim ui64 As System.UInt64  ' corresponding .NET type

'-----------------------------
' const values
'-----------------------------
Const GÜTERSLOH As UShort = 33333
Const MOON As UInt32 = 384400UI
Const SUN As UInt32 = 149600000UI
Const LICHTJAHR As ULong = 9460730472580UL
```

Value types, Integral types

```
'-----------------------------
' literals
'-----------------------------

'   +-------------------------+-----------+------------+
'   | Literal type character | Data type |  Example   |
'   +-------------------------+-----------+------------+
'   | S                       | Short     | i = 347S   |
'   | I                       | Integer   | i = 347I   |
'   | L                       | Long      | i = 347L   |
'   | D                       | Decimal   | i = 347D   |
'   | F                       | Single    | i = 347F   |
'   | R                       | Double    | i = 347R   |
'   | US                      | UShort    | i = 347US  |
'   | UI                      | UInteger  | i = 347UI  |
'   | UL                      | ULong     | i = 347UL  |
'   | C                       | Char      | i = "."C   |
'   +-------------------------+-----------+------------+
```

Value types, Integral types

Datentyp	Umschließendes Zeichen	Angehängtes Typzeichen
Boolean	(kein)	(kein)
Byte	(kein)	(kein)
Char	"	C
Date	#	(kein)
Decimal	(kein)	D oder @
Double	(kein)	R oder #
Integer	(kein)	I oder %
Long	(kein)	L oder &
Short	(kein)	S
Single	(kein)	F oder !
String	"	(kein)

Value types, Integral types

```
i = &O1234567 ' octal 0-7
j = &HCAFFEE ' hexadecimal 0-F

System.Console.WriteLine(i) ' 342391
System.Console.WriteLine(j) ' 13303790

Const BLUE As Integer = &HFF
System.Console.WriteLine("RGB BLUE={0:X6}", BLUE) ' 0000FF

' input an octal value
Dim octalNumber% = Global.Microsoft.VisualBasic.Val(
    "&O" + Global.Microsoft.VisualBasic.InputBox("input octal number"))
System.Console.WriteLine("decimal value=" & octalNumber)

' input a binary value
Dim binaryNumber As String = InputBox("input binary number")
Dim binaryValue As ULong
For i = 1 To binaryNumber.Length
    binaryValue = binaryValue + Val(
        Global.Microsoft.VisualBasic.Mid(
            binaryNumber, binaryNumber.Length - i + 1, 1)) *
                Global.System.Math.Pow(2, i - 1)
Next
System.Console.WriteLine("binary number {0}={1}(10)",
                         binaryNumber,
                         binaryValue)
```

Value types, Integral types

```
' convert decimal to binary
' variant 1 (toString method)
i = 200
System.Console.WriteLine(System.Convert.ToString(i, 2)) ' 11001000

' variant 2 (consecutive division)
binaryNumber = String.Empty
Do
    binaryNumber = i Mod 2 & binaryNumber
    i = i \ 2
Loop While CBool(i)
System.Console.WriteLine(binaryNumber) ' 11001000

' convert decimal to hexadecimal
' variant 1 (hex function)
Dim strHex As String = Global.Microsoft.VisualBasic.Hex(13303790)
System.Console.WriteLine(strHex) ' CAFFEE

' variant 2 (convert class)
strHex = System.Convert.ToString(13303790, 16)
System.Console.WriteLine(strHex.ToUpper) ' CAFFEE

' variant 3 (format method)
System.Console.WriteLine(String.Format("{0:X}", 13303790)) ' CAFFEE
```

Value types, Integral types

```
'-----------------------------
' identifier types
'-----------------------------

'   +---------------------------+-----------+--------------------+
'   | Identifier type character | Data type |      Example       |
'   +---------------------------+-----------+--------------------+
'   | %                         | Integer   | Dim L%             |
'   | &                         | Long      | Dim M&             |
'   | @                         | Decimal   | Const W@ = 37.5    |
'   | !                         | Single    | Dim Q!             |
'   | #                         | Double    | Dim X#             |
'   | $                         | String    | Dim V$ = "Secret"  |
'   +---------------------------+-----------+--------------------+

'-----------------------------
' enclosing/type characters
'-----------------------------

'   +-----------+---------------------+-------------------------+
'   | Data type | Enclosing character | Appended type character |
'   +-----------+---------------------+-------------------------+
'   | Boolean   | (none)              | (none)                  |
'   | Byte      | (none)              | (none)                  |
'   | Char      | "                   | C                       |
'   | Date      | #                   | (none)                  |
'   | Decimal   | (none)              | D or @                  |
'   | Double    | (none)              | R or #                  |
'   | Integer   | (none)              | I or %                  |
'   | Long      | (none)              | L or &                  |
'   | Short     | (none)              | S                       |
'   | Single    | (none)              | F or !                  |
'   | String    | "                   | (none)                  |
'   +-----------+---------------------+-------------------------+
```

Value types, Integral types

```
' force type char
Const ch As Char = "A"c

' DateTime constants
Const [date] As DateTime = #1/11/2011#
Const [time] As DateTime = #1:10:20 AM#

' force type long
Const lng As Long = 45L

' force type single
Const sng As Single = 45.55!
```

Value types, Integral types

```
'-----------------------------
' default values
'-----------------------------
Dim charVal As Char, intVal As Integer, singleVal As Single,
    decimalVal As Decimal, boolVal As Boolean,
    dateTimeVal As System.DateTime, strVal As System.String,
    varTypeVal As VariantType, varVal
System.Console.WriteLine(charVal)     '\0
System.Console.WriteLine(intVal)      ' 0
System.Console.WriteLine(singleVal)   ' 0
System.Console.WriteLine(decimalVal)  ' 0
System.Console.WriteLine(boolVal)     ' false
System.Console.WriteLine(dateTimeVal) ' 01.01.0001 00:00:00
System.Console.WriteLine(strVal)      ' "", warning not assigned
System.Console.WriteLine(varTypeVal)  ' 0
System.Console.WriteLine(varVal)      ' warning not assigned

varVal += 1
System.Console.WriteLine(varVal)      ' 1
```

Value types, Integral types

```
'-----------------------------
' integral promotion
'-----------------------------
'   Objects of an integral type can be converted to another wider integral
'   type (that is, a type that can represent a larger set of values).
'   This widening type of conversion is called integral promotion.
i16 = by ' by (byte) is widened to an integer
by = i16 ' ok, but may result in runtime error

i16 = 256
' by = i16                          ' runtime error
' by = System.Convert.ToByte(i16) ' runtime error

' i16 = ch; ' error, no implicit conversion
i16 = Global.Microsoft.VisualBasic.Asc(ch) ' ok
System.Console.WriteLine(i16) ' 65
i16 = Global.Microsoft.VisualBasic.Val(ch) ' ok
System.Console.WriteLine(i16) ' 0
i16 = Global.Microsoft.VisualBasic.Val("1"c) ' ok
System.Console.WriteLine(i16) ' 1

i32 = 1.1R ' implicit conversion, but data gets lost
System.Console.WriteLine(i32) ' 1
i32 = 3 / 4 ' 0.75 round up
System.Console.WriteLine(i32) ' 1
```

Value types, Integral types

```
        i32 = 1 << 31
        System.Console.WriteLine(i32)              ' -2147483648
        System.Console.WriteLine(Integer.MinValue) ' -2147483648

        i32 = 1 << 32 ' 1 (cyclic shifting)
        i32 = 1 << 33 ' 2

        System.Console.WriteLine(1L << 63)  ' -9223372036854775808
        System.Console.WriteLine(1UL << 63) '  9223372036854775808

        i64 = 10000000000
        Try
            by = i64 ' runtime error, OverflowException
        Catch
            System.Console.WriteLine("overflow")
        End Try
    End Sub
End Module
```

Value types, Floating point types

```
IEEE 754
---------------------------------
32 Bits |S 31|E 30 - 23|M 22 - 0|
---------------------------------
Sign 1 Bit: 0+ 1-
Exponent 8 Bits
Mantissa 23 Bits
Form: ±(d1.d2d3d4...dn) x 2^e
Ex.: 0.10101(2) x 2^-1 = (0 x 2^0 + 1 x 2^-1 + ... + 1 x 2^-5) x 2^-1 = 0.328125(10)

Normalization: force the integer part of the mantissa to be exactly 1
=> 1.0101(2) x 2^-2

For neg. Exponent: subtract 127, Ex.: 2^-2, -2 = 125-127 ≙ 01111101 (=125)
=> 0 01111101 x 1.0101, 1 predecimal won't be saved => 0 01111101 01010000000000000000000 (32 Bit)

Ex.: 0.09375(10) = 2^-4 + 2^-5, M: 00011000000000000000000 (23 Bit)
M: 1.10000000000000000000000 (normalized, 4 x shift left)
=> Exp.: 4 x shift right
E: 2^-4 = 01111011 (=123-127)
=> 01111011 x 1.10000000000000000000000
=> 0 01111011 x 1.10000000000000000000000 with Signbit
=> 0 01111011 x   10000000000000000000000 (32 Bit)
=> 00111101110000000000000000000000

Online Calculator IEEE-754 Floating-Point Conversion:
http://babbage.cs.qc.cuny.edu/IEEE-754.old/Decimal.html

The IEEE standard has four different rounding modes
• Round to Nearest (default) – rounds to the nearest value; if the number
  falls midway it is rounded to the nearest value with an even (zero) least
  significant bit, which occurs 50% of the time
• Round toward 0 – directed rounding towards zero
• Round toward +∞ – directed rounding towards positive infinity
• Round toward -∞ – directed rounding towards negative infinity
```

Value types, Floating point types

```
Module BasicDatatypes
    Sub floatingPointTypes()
        Dim f As Single ' 32bit, ±1.5 × 10^−45 to ±3.4 × 10^38, 7 digits
        f = 1.0 ' double literal
        f = 1.0F
        f = 1.0E+10F
        f = 1.0E-10F
        f = Single.NaN
        f = Single.NegativeInfinity ' very small numbers
        f = Single.PositiveInfinity ' very large numbers
        System.Console.WriteLine(f * 1) ' +unendlich
        System.Console.WriteLine(System.Math.Pow(2, 2000)) ' +unendlich

        System.Console.WriteLine(Single.Epsilon) ' smallest positive value > 0
        System.Console.WriteLine(Single.MinValue)
        System.Console.WriteLine(Single.MaxValue)
        ' or
        System.Console.WriteLine(System.Single.Epsilon)
        System.Console.WriteLine(System.Single.MinValue)
        System.Console.WriteLine(System.Single.MaxValue)

        Dim d As Double ' 64bit, ±5.0 × 10^−324 to ±1.7 × 10^308, 15-16 digits
        d = 1.0R
        System.Console.WriteLine(System.Double.Epsilon)
        System.Console.WriteLine(System.Double.MinValue)
        System.Console.WriteLine(System.Double.MaxValue)
```

Value types, Floating point types

```
        ' automatic promotion
        d = f + d ' infinity
        f = f + d ' infinity
        d = 2D * 1.0E+30F ' but loss of precision, literal itself is imprecisely
        System.Console.WriteLine(d) ' = 2,00000003009493E+30

        '----------------------------
        ' Rounding error
        '----------------------------
        ' Loss of significance and inaccurate calculations
        System.Console.WriteLine("double calculation: " & 100000000.0 - 1) ' = 99999999
        System.Console.WriteLine("single calculation: " & 1.0E+8F - 1)     ' = 1E+08

        ' infinite loop
#If True Then
        Dim l As Long = 0
        For f = 0 To 100000000.0
            l += 1
            If l >= 100000000.0 Then
                System.Console.WriteLine("{0:N1} {1:F} loop still busy", l, f)
            End If
        Next

#End If
```

Value types, Floating point types

```
System.Console.WriteLine("{0:F15}", 1.1 * 1.1)           ' = 1,210000000000000
System.Console.WriteLine("{0:F15}", 1.1R * 1.1)          ' = 1,210000000000000
System.Console.WriteLine("{0:F15}", 1.1F * 1.1)          ' = 1,210000026226040
System.Console.WriteLine("{0:F15}", 1.1 * 1.1F)          ' = 1,210000026226040
System.Console.WriteLine("{0:F15}", 1.1F * 1.1F)         ' = 1,210000000000000
System.Console.WriteLine("{0:F15}", 1190 * 0.19F / 1.19) ' = 189,999992306493000
System.Console.WriteLine("{0:F15}", 1190 * 0.19 / 1.19)  ' = 190,000000000000000

' rounding error in associativity
System.Console.WriteLine(1.0E+10F * (1.0E+10F - 1.0E+10F + 1.0E-10F)) ' = 1
' calculation error when adjusting exponents
System.Console.WriteLine(1.0E+10F * (1.0E+10F + 1.0E-10F - 1.0E+10F)) ' = 0,

System.Console.WriteLine(
    10000000000.0 * (10000000000.0 + 0.0000000001 - 10000000000.0))     ' = 0
System.Console.WriteLine(
    10000000000.0R * (10000000000.0R + 0.0000000001R - 10000000000.0R)) ' = 0
System.Console.WriteLine(
    10000000000D * (10000000000D + 0.0000000001D - 10000000000D))       ' = 1,
                                                                        ' now correct
```

Value types, Floating point types

```
System.Console.WriteLine(1000000 + 0.0000001F)    ' = 1000000
System.Console.WriteLine(1000000.0F + 0.0000001F) ' = 1000000
System.Console.WriteLine(1000000 + 0.000001F)     ' = 1000000
System.Console.WriteLine(1000000 + 0.00001F)      ' = 1000000
System.Console.WriteLine(1000000 + 0.0001F)       ' = 1000000
System.Console.WriteLine(1000000 + 0.001F)        ' = 1000000
System.Console.WriteLine(1000000 + 0.01F)         ' = 1000000
System.Console.WriteLine(1000000.0F + 0.1F)       ' = 1000000
System.Console.WriteLine(1000000 + 1.0F)          ' = 1000001

System.Console.WriteLine(1000000 + 0.0000001)  ' = 1000000,0000001
System.Console.WriteLine(1000000 + 0.000001)   ' = 1000000,000001
System.Console.WriteLine(1000000 + 0.00001)    ' = 1000000,00001
System.Console.WriteLine(1000000 + 0.0001)     ' = 1000000,0001
System.Console.WriteLine(1000000 + 0.001)      ' = 1000000,001
System.Console.WriteLine(1000000 + 0.01)       ' = 1000000,01
System.Console.WriteLine(1000000 + 0.1)        ' = 1000000,1
System.Console.WriteLine(1000000 + 1)          ' = 1000001
```

Value types, Floating point types

```
        ' implicit cast
        Dim i As Integer
        f = 1UL
        i = f

        ' explicit cast
        i = System.Convert.ToInt32(1.0F)

        ' Ariane V, 4.Juni 96, 36.7s self destruction after take-off
        ' Reason: Conversion float(64) to int(16)
        i = 9999.999R ' = 10000
        System.Console.WriteLine(i)

        ' breaking a floating point number into its binary significand (a floating point
        ' mantissa with an absolute value 0.5 ≤ m < 1, 0.00..01 → 0.10..00) and an in-
        ' tegral exponent for 2
        f = -13.5F
        Dim i64 As System.Int64 = System.BitConverter.DoubleToInt64Bits(f)
        System.Console.WriteLine(System.Convert.ToString(i64, 2))
        ' = 1 10000000010 1011000000000000000000000000000000000000000000000
    End Sub
End Module
```

Übung - IEEE-754 Floating-Point Conversion

a) Berechnen Sie die floating-point Darstellung der Zahl 13.5
Verwenden Sie bei Bedarf die folgende Tabelle:

IEEE 754, Mantisse 24 Bit

Bit	Exponent	Wert	Wert dezimal
1	-1	1/2	0,500000000000000000000000
2	-2	1/4	0,250000000000000000000000
3	-3	1/8	0,125000000000000000000000
4	-4	1/16	0,062500000000000000000000
5	-5	1/32	0,031250000000000000000000
6	-6	1/64	0,015625000000000000000000
7	-7	1/128	0,007812500000000000000000
8	-8	1/256	0,003906250000000000000000
9	-9	1/512	0,001953125000000000000000
10	-10	1/1024	0,000976562500000000000000

Lösung - IEEE-754 Floating-Point Conversion

a)

```
13.5 ≙ 1.ddddd x 2^n → 13.5 = 1.6875 x 2^3
Exponent: 130-127=3 (≙ 2^3), 130(10) = 10000010(2)
Mantissa: 1.6875(10) = 1.10110000000000000000000(2)
Signed  : 0
→ 13.5(10) = 0_10000010_10110000000000000000000 = 01000001010110000000000000000000(2)
```

Value types

```
Module BasicDatatypes
    Sub valueTypes()
        '-----------------------------
        ' boolean
        '-----------------------------
        Dim b As Boolean = True ' default False
        ' equivalent to
        Dim bool As New Global.System.Boolean()

        ' corresponding .NET Framework type: System.Boolean structure
        Dim bStruct As System.Boolean
        Global.System.Diagnostics.Debug.Print(b.GetType.ToString) ' System.Boolean
        Debug.Print(bStruct.GetType.ToString) ' System.Boolean
        Debug.Print(b.CompareTo(bStruct))     ' 0, equal

        Dim ready, found As Boolean

        ready = found = False ' ↔ ready = (found = False), assignment of a comparison
        Debug.Print(ready) ' True
        Debug.Print(found) ' False
        Debug.Print(ready.CompareTo(found)) ' 1, >0, not equal

        ready = found = True
        Debug.Print(ready.CompareTo(found)) ' 0, equal

        ready = True
        found = False
        Debug.Print(ready.CompareTo(found)) ' 1, >0, not equal

        ready = False
        found = True
        Debug.Print(ready.CompareTo(found)) ' -1, <0, not equal
```

Value types

```
' automatic promotion
bool = 22
Debug.Print(bool) ' True (<>0 ↔ True)

bool = 0
Debug.Print(bool) ' False

' bool = "yes" ' System.InvalidCastException
' bool = "no"  ' System.InvalidCastException

bool = False * True
Debug.Print(bool) ' False

bool = True * -1
Debug.Print(bool) ' True
Debug.Print(CInt(bool)) ' -1

Dim ib As Integer
ib = True * -1
Debug.Print(ib) ' 1

Debug.Print(1 > 5 And 3 - 2 > 1 Or 4 <> 3) ' True
' =
Debug.Print(((1 > 5) And ((3 - 2) > 1)) Or (4 <> 3))
```

Value types

```
' parsing
bool = Boolean.Parse("true")
System.Console.WriteLine(bool) ' True
' bool = Boolean.Parse("1") ' runtime error

' converting cbool, inlined
bool = CBool(1)      ' True
bool = CBool(1.0)    ' True
bool = CBool(-1)     ' True
bool = CBool(-10)    ' True
bool = CBool(1D)     ' True
bool = CBool(0)      ' False
bool = CBool("1")    ' True
bool = CBool("True") ' True
' bool = CBool("Yes")  ' error

' converting convert class
Dim strBool As String = bool.ToString()
bool = System.Convert.ToBoolean(123)
' bool = System.Convert.ToBoolean("123") ' error
' bool = System.Convert.ToBoolean("-1")  ' error
bool = System.Convert.ToBoolean("False")
bool = System.Convert.ToBoolean("tRuE")
' bool = System.Convert.ToBoolean("No")  ' error
bool = System.Convert.ToBoolean(1.23)
' bool = System.Convert.ToBoolean(#1/1/2010#) ' error
bool = System.Convert.ToBoolean(123D)
```

Value types

```
' customized output
Dim bigCustomer As Boolean = True
System.Console.WriteLine("you get " +
                         If(bigCustomer, "20%", "5%") + " discount")

'----------------------------
' decimal
'----------------------------
' The binary representation of a Decimal value consists of a 1-bit sign,
' a 96-bit integer number, and a scaling factor used to divide the 96-bit
' integer and specify what portion of it is a decimal fraction. The scaling
' factor is implicitly the number 10, raised to an exponent ranging from 0
' to 28.
' • not a floating-point data type
' + highest precision, up to 29 significant digits
' - performance, slowest of all numeric types

Dim de As Decimal
de = 0D
de = 1D
System.Console.WriteLine(Decimal.MaxValue) ' 79228162514264337593543950335
System.Console.WriteLine(Decimal.MinValue) '-79228162514264337593543950335

System.Console.WriteLine(Decimal.op_Increment(de)) ' 2
de += 1
```

Value types

```
' GetBits, returns an integer array with four elements.
' The first, second, and third elements of the returned array contain the
' low, middle, and high 32 bits of the 96-bit integer number, the fourth
' element of the returned array contains the scale factor and sign.
' Forth element: bits 0 to 15 unused and must be zero, bits 16 to 23 must
' contain an exponent between 0 and 28, which indicates the power of 10
' to divide the integer number, bits 24 to 30 are unused and must be zero,
' bit 31 contains the sign: 0 mean positive, 1 means negative.
Dim iArr(4) As Integer ' → arrays
de = -65535D
iArr(0) = System.Decimal.GetBits(de)(0)
iArr(1) = System.Decimal.GetBits(de)(1)
iArr(2) = System.Decimal.GetBits(de)(2)
iArr(3) = System.Decimal.GetBits(de)(3)
System.Console.WriteLine(Global.Microsoft.VisualBasic.Constants.vbCrLf +
    "{0:X} {1}", iArr(0), System.Convert.ToString(iArr(0), 2))
System.Console.WriteLine(
    "{0:X} {1}", iArr(1), System.Convert.ToString(iArr(1), 2))
System.Console.WriteLine(
    "{0:X} {1}", iArr(2), System.Convert.ToString(iArr(2), 2))
System.Console.WriteLine(
    "{0:X} {1}", iArr(3), System.Convert.ToString(iArr(3), 2))
System.Console.WriteLine(
    "Bit 31 is " + If(iArr(3) And (1 << 31), "set (neg.)", "not set (pos.)"))
```

Value types

```
de = 255D
iArr(0) = System.Decimal.GetBits(de)(0)
iArr(1) = System.Decimal.GetBits(de)(1)
iArr(2) = System.Decimal.GetBits(de)(2)
iArr(3) = System.Decimal.GetBits(de)(3)
System.Console.WriteLine(vbCrLf +
    "{0:X} {1}", iArr(0), System.Convert.ToString(iArr(0), 2))
System.Console.WriteLine(
    "{0:X} {1}", iArr(1), System.Convert.ToString(iArr(1), 2))
System.Console.WriteLine(
    "{0:X} {1}", iArr(2), System.Convert.ToString(iArr(2), 2))
System.Console.WriteLine(
    "{0:X} {1}", iArr(3), System.Convert.ToString(iArr(3), 2))
System.Console.WriteLine(
    "Bit 31 is " + If(iArr(3) And (1 << 31), "set (neg.)", "not set (pos.)"))

' constructing a decimal
Dim lo, mid, hi As Integer, isNegative As Boolean, scale As Byte
lo = 0
mid = 100
hi = 1000
isNegative = True
scale = 3 ' decimal places
Dim conDec As New Decimal(lo, mid, hi, isNegative, scale)
System.Console.WriteLine(
    "constructed decimal=" & conDec) ' -18446744074139048345,600
```

Übung - Einfache Datentypen

a) Definieren Sie für die folgenden Wertebereiche adäquate Variablen bzw. Konstanten.
Achten Sie darauf, den kleinstmöglichen Datentyp festzulegen.

Bezeichner	Wertebereich, Wert
Stückzahl	[0..1Mio]
ArtikelNummer	[1..10000]
Rating	["A"]
Smiley	["☺"], Codepage 850, \001
colorIndex	[0..10]
BetriebsZustand	[on/off]
Elementarladung	[1,60217733·10^-19]
MilchstraßeØ	[700000000000000000]
RGB_Red	[FF0000]
Kostenfaktor	[35%]
MountEverest	[8848]
MilesToKm	[1,60934]
SerialPort16bit	[1100110011110000]

Eingabe

```
Console.WriteLine("About to call Console.ReadLine in a loop.")
Console.WriteLine("----")
Dim s As String
Dim ctr As Integer
Do
ctr += 1
s = Console.ReadLine()
Console.WriteLine("Line {0}: {1}", ctr, s)
Loop While s IsNot Nothing
Console.WriteLine("---")

Dim array(4) As Char
array(0) = "d"c
array(1) = "n"c
array(2) = "p"c
array(3) = "x"c
array(4) = "x"c
Console.WriteLine(array, 0, 3)
```

Eingabe

```
While True

' Read value.
Dim s As String = Console.ReadLine()

' Test the value.
If s = "1" Then
Console.WriteLine("One")
ElseIf s = "2" Then
Console.WriteLine("Two")
End If

' Write the value.
Console.WriteLine("You typed " + s)

End While
```

Eingabe

```
Sub Main()
    ' Set Foreground and Background colors.
    ' ... You can just set one.
    Console.ForegroundColor = ConsoleColor.Red
    Console.BackgroundColor = ConsoleColor.DarkCyan

    Console.WriteLine("Warning")
    Console.WriteLine("There is a disturbance in the Force")

    ' Reset the colors.
    Console.ResetColor()

    Console.WriteLine("Sorry")
End Sub
```

Eingabe

```
Sub console_test()
    Dim c As System.Console ' Intrinsic Object
    ' --- Einfache Ausgaben
    c.WriteLine("Zeile1")
    c.Out.WriteLine("Zeile2")
    c.Out.Write("Zeile...")
    c.Write("3!")
    c.WriteLine()
    Dim o As Object
    c.WriteLine(o)

    ' --- zeilenweise Eingabe lesen
    c.WriteLine("Ihr Vorname?")
    Dim vorname As String
    vorname = c.ReadLine()
    Dim nachname As String
    c.WriteLine("Ihr Nachname?")
    nachname = c.ReadLine()

    ' --- Ausgabe mit Platzhaltern
    c.WriteLine("Hallo {0} {1}!", vorname, nachname)

    ' -- Warten bis ein e eingegeben wurde!
    Dim eingabe As Integer
    While True
        eingabe = c.Read
        ' Lesen, bis "e" eingegeben wird
        If eingabe = Asc("e") Then Exit While
        c.WriteLine(eingabe)
    End While
```

Eingabe

```
Console.WriteLine( &
        "Enter the order date and time (mm/dd/yyyy hh:mm AM/PM)")
OrderDate = CDate(Console.ReadLine())

' Request the quantity of each category of items
Console.Write("Number of Shirts:  ")
Dim strShirts As String = Console.ReadLine()
NumberOfShirts = CInt(strShirts)

Console.Write("Number of Pants:   ")
Dim strPants As String = Console.ReadLine()
NumberOfPants = CInt(strPants)

Console.Write("Number of Dresses: ")
Dim strDresses As String = Console.ReadLine()
NumberOfDresses = CInt(strDresses)

' Perform the necessary calculations
SubTotalShirts = NumberOfShirts * PriceOneShirt
SubTotalPants = NumberOfPants * PriceAPairOfPants
SubTotalDresses = NumberOfDresses * PriceOneDress
' Calculate the "temporary" total of the order
TotalOrder = SubTotalShirts + SubTotalPants + SubTotalDresses

' Calculate the tax amount using a constant rate
TaxAmount = TotalOrder * TaxRate
' Add the tax amount to the total order
SalesTotal = TotalOrder + TaxAmount

' Communicate the total to the user...
Console.Write("\nThe Total order is: ")
Console.WriteLine(SalesTotal)
' and request money for the order
Console.Write("Amount Tended? ")
AmountTended = CDbl(Console.ReadLine())

' Calculate the difference owed to the customer
' or that the customer still owes to the store
Difference = AmountTended - SalesTotal
Console.WriteLine()

' Display the receipt
Console.WriteLine("===================================")
Console.WriteLine("-/- Georgetown Cleaning Services -/-")
Console.WriteLine("===================================")
Console.WriteLine("Customer:   " + CustomerName)
Console.WriteLine("Home Phone: " + HomePhone)
Console.WriteLine("Date & Time: " + CStr(OrderDate))
```

Prüfen

```
Function sphericalVolume(ByVal sngRadius As Single) As Double 'Rückgabe als Double/Variant, da
                                                              'sonst ungenau
    sphericalVolume = Round(4 / 3 * WorksheetFunction.Pi() * sngRadius ^ 3, 4)
End Function

'Problem: Übergabe Leerzelle (Empty), Strings, Datumswerte etc.

Function sphericalVolume(ByVal vntRadius As Object) As Object
    Select Case VarType(vntRadius)
        Case vbEmpty:  sphericalVolume = "Leerzelle unzulässig"
        Case vbString: sphericalVolume = "Text unzulässig"
        Case vbDate:   sphericalVolume = "Datum unzulässig"
        Case Else:     sphericalVolume = Round(4 / 3 * WorksheetFunction.Pi() * vntRadius ^ 3, 4)
    End Select
End Function
```

ByRef

```
Sub Kontodeckung(ByVal Kontostand, Betrag)
    'Prüfung, ob Abbuchung möglich
    Kontostand = Kontostand – Betrag
    If Kontostand >= 0 Then
        Call AbschlussTransaktion
    Else
        Call AbbruchTransaktion
    End If
End Sub

Sub Abbuchung()
    AktuellerKontostand = 500
    Call Kontodeckung(AktuellerKontostand, 1000)
    Debug.Print AktuellerKontostand
End Sub
```

Übung - Einfache Datentypen

b) Welche Werte haben die Variablen im folgenden Programmstück und wie viele Bytes werden von ihnen jeweils belegt?

```
Sub test()
    Dim a, b, c As Integer
    Dim d, e, f As Long
    Dim g, h, i As Single
    Dim j, k, l As Double
    Dim m, n, o, p As SByte

    a = 1.0E+10F * (1.0E+10F + 1.0E-10F - 1.0E+10F)
    a = 3.75
    b = 2 / 3
    d = 100000L * 100000
    c = CInt(d)
    c = System.Convert.ToInt32(d)
    c = CType(d, Integer)
    g = 1 / 3
    h = 1D / 3D
    j = 1 / 3
    b = j
    m = 127
    n = m + 1
    n = System.Convert.ToSByte(m + 1)
    o = 255
    p = -150
    p = h
End Sub
```

Übung - Einfache Datentypen

c) Transformieren Sie die folgenden mathematischen Formeln in gültige VB-Ausdrücke.
Verwenden Sie hierzu die System.Math-Klasse und die folgenden Funktionen:

- Die Wurzel eines Ausdrucks kann dann mittels der Sqrt-Funktion berechnet werden:
  Sqrt(4) = 2
- Die Potenz eines Ausdrucks:
  Pow(2,3) = 2^3 = 8
- Der Betrag |a| eines Ausdrucks:
  Abs(-1) = 1
- Die Exponentialfunktion:
  Exp(2) = e^2
- π, Pi

I. f(A, B) = $\frac{|A-B|}{|A|+|B|}$

II. f(C, L, R) = $\sqrt{\frac{1}{LC}-\left(\frac{R}{2L}\right)^2}$

III. f(N) = $e^{-N}N^N\sqrt{2N\pi}$

Übung - Einfache Datentypen

d) Überprüfen Sie die folgenden Ausdrücke auf Richtigkeit.

```
Sub test()
    Const q As Integer = 1
    Const r As Integer = 7.77
    Const s As Double
    Const t As Single = 1000UI
    Const u As UInt32 = 333L
    Const v As Char = 1
    Const fire As Short = 112
    Const high As Short = 1
    Const low As Short = OFF
    Const w As Char = "1"
    Const error As Char = "error"
    Const x As Integer = "2"
    Const y As Single = 1.0F
    Const z As Integer = w + x
    Const four As Char = "f" - "a"
    Const A As Integer = x + y
    Const B As Integer = 0
    Dim C As Integer = 1
    Const D As Single = C
End Sub
```

Lösung - Einfache Datentypen

```
Module DataTypesEx
    Sub Main()
        ' a)
        Dim Stückzahl As UInt32 = 0
        Dim ArtikelNummer As UShort = 555
        Dim Rating As Char = "A"c
        Rating = CChar("A")
        Dim Smiley As Char = System.Convert.ToChar(1)
        Smiley = Global.Microsoft.VisualBasic.Chr(1)
        Dim colorIndex As SByte = 0
        Dim BetriebsZustand As Boolean = True
        Const [ON] As Byte = 1
        Const OFF As Byte = 0
        Dim yBetriebsZustand As Byte = [ON]
        Const Elementarladung As Decimal = 0.00000000000000000160217733D
        Dim MilchstraßeØ As ULong = 700000000000000000UL
        MilchstraßeØ = System.Convert.ToInt64(7.0E+17)
        MilchstraßeØ = CULng(7.0E+17)
        Const RGB_Red As UInt32 = &HFF0000
        Const Kostenfaktor As Single = 0.35F
        Const MountEverest As UShort = 8848
        Const MilesToKm As Double = 1.60934
        Dim SerialPort16bit As System.UInt16 = System.Convert.ToUInt16("1100110011110000", 2)

        System.Console.WriteLine(Stückzahl)
        System.Console.WriteLine(ArtikelNummer)
        System.Console.WriteLine(Rating)
        System.Console.WriteLine(Smiley)
        System.Console.WriteLine(colorIndex)
        System.Console.WriteLine(BetriebsZustand)
        System.Console.WriteLine(yBetriebsZustand)
        System.Console.WriteLine(Elementarladung)
        System.Console.WriteLine(MilchstraßeØ)
        System.Console.WriteLine(RGB_Red)
        System.Console.WriteLine(Kostenfaktor)
        System.Console.WriteLine(MountEverest)
        System.Console.WriteLine(MilesToKm)
        System.Console.WriteLine(SerialPort16bit)
```

Lösung - Einfache Datentypen

```
' b)
Dim a, b, c As Integer
Dim d, e, f As Long
Dim g, h, i As Single
Dim j, k, l As Double
Dim m, n, o, p As SByte

System.Console.WriteLine("Size of Integer: " &
    System.Runtime.InteropServices.Marshal.SizeOf(a) & " Bytes") ' = 4
System.Console.WriteLine("Size of Long: " &
    System.Runtime.InteropServices.Marshal.SizeOf(d) & " Bytes") ' = 8
System.Console.WriteLine("Size of Single: " &
    System.Runtime.InteropServices.Marshal.SizeOf(g) & " Bytes") ' = 4
System.Console.WriteLine("Size of Double: " &
    System.Runtime.InteropServices.Marshal.SizeOf(j) & " Bytes") ' = 8
System.Console.WriteLine("Size of SByte: " &
    System.Runtime.InteropServices.Marshal.SizeOf(m) & " Bytes") ' = 1

a = 1.0E+10F * (1.0E+10F + 1.0E-10F - 1.0E+10F) ' = 0
a = 3.75  ' = 4
b = 2 / 3 ' = 1
d = 100000L * 100000 ' = 10.000.000.000
' c = CInt(d) ' error, overflow
' c = System.Convert.ToInt32(d) ' error, overflow
' c = CType(d, Integer)
c = Bin_To_Dec(
        Global.Microsoft.VisualBasic.Right(' convert only 32 bit
            System.Convert.ToString(d, 2), 32)) ' = 1410065408
' or
c = Convert.ToInt32(Right(Convert.ToString(d, 2), 32), 2)
g = 1 / 3   ' = 0,333...43
h = 1D / 3D ' = 0,333...43
j = 1 / 3   ' = 0.333...31
b = j   ' = 0
m = 127 ' = 127
' n = m + 1 ' error, overflow
' n = System.Convert.ToSByte(m + 1) ' error, overflow
Try
    n = System.Convert.ToSByte(m + 1)
Catch
    n = n.GetType().GetField("MaxValue").GetValue(Nothing)
    ' or
    n = If(TypeName(n) = "SByte", SByte.MaxValue, 0)
    ' VarType has no SByte enumeration member
End Try
' o = 255 ' error, overflow
' p = -150' error, overflow
p = h ' = 0
```

Lösung - Einfache Datentypen

```
    ' c)
    System.Console.WriteLine(System.Math.Sqrt(4))   ' = 2
    System.Console.WriteLine(System.Math.Pow(2, 3)) ' = 8
    System.Console.WriteLine(System.Math.Abs(-1))   ' = 1
    System.Console.WriteLine(System.Math.Exp(2))    ' = 7,38905609893065
    System.Console.WriteLine(System.Math.PI)        ' = 3,14159265358979

    Dim _A, _B, _L, _C, _R, _N
    Dim f_i = System.Math.Abs(_A - _B) / (System.Math.Abs(_A) + System.Math.Abs(_B))
    Dim f_ii = System.Math.Sqrt(1 / (_L - _C) - System.Math.Pow(_R / (2 * _L), 2))
    Dim f_iii = System.Math.Exp(-_N) *
                System.Math.Pow(_N, _N) *
                System.Math.Sqrt(2 * _N * System.Math.PI)

    ' d)
    Const q As Integer = 1
    Const r As Integer = 7.77 ' = 8
    ' Const s As Double ' error, missing initialization
    Const s As Double = 0
    Const t As Single = 1000UI
    Const u As UInt32 = 333L
    ' Const v As Char = 1 ' error, no implicit conversion
    ' Const v As Char = System.Convert.ToChar(1) ' error, not constant
    ' Const v As Char = CChar(1) ' error, can't be converted
    Const v As Char = Global.Microsoft.VisualBasic.Chr(1)
    Const fire As Short = 112
    Const high As Short = 1
    Const low As Short = OFF
    Const w As Char = "1"c
    ' Const error As Char = "error" ' error, keyword
    Const [error] As Char = "error"
    ' Const x As Integer = "2" ' error, no conversion from string to integer
    ' Const x As Integer = Integer.Parse("2") ' error, not
    Const x As Integer = Global.Microsoft.VisualBasic.Asc("2")
    Const y As Single = 1.0F
    Const z As Integer = Asc(w) + x
    ' Const four As Char = "f" - "a" ' error no conversion from double to char
    Const four As Char = Chr(Asc("f") - Asc("a"))
    Const A_ As Integer = x + y
    ' B_ const As Integer = 0 ' error
    Const B_ As Integer = 0
    ' Dim C_ As Integer = 1 ' error prone, C_ is not const → 'Failed to emit module'
    Const C_ As Integer = 1
    Const D_ As Single = C_
End Sub
```

Lösung - Einfache Datentypen

```
    Function Bin_To_Dec(ByVal Bin As String)
        Dim dec As Double = Nothing
        Dim length As Integer = Len(Bin)
        Dim temp As Integer = Nothing
        Dim x As Integer = Nothing
        For x = 1 To length
            temp = Val(Mid(Bin, length, 1))
            length = length - 1
            If temp <> "0" Then
                dec += (2 ^ (x - 1))
            End If
        Next
        Return dec
    End Function
End Module
```

Date and Time

```
Imports System.Console

Module DateAndTime
    Sub Main()
        REM VB
        WriteLine(Microsoft.VisualBasic.DateAndTime.TimeString) ' Time
        WriteLine(Microsoft.VisualBasic.DateAndTime.TimeOfDay)  ' from 1.1.0001

        WriteLine(Microsoft.VisualBasic.DateAndTime.TimeSerial(50, 50, 30))
        ' from 1.1.0001

        WriteLine(Microsoft.VisualBasic.DateAndTime.Now) ' Date + Time

        WriteLine(Microsoft.VisualBasic.DateAndTime.Timer)
        ' 55951,4531196, s since midnight

        REM .NET
        WriteLine(System.DateTime.Today) ' Date, midnight
        WriteLine(System.DateTime.Now)   ' Date + Time
    End Sub
End Module
```

Value types

```
        '-----------------------------
        ' datetime structure
        '-----------------------------
        Dim dateVal As System.DateTime = System.DateTime.Today
        Dim timeVal As System.DateTime = System.DateTime.Now
        System.Console.WriteLine("Date: " + dateVal) ' date
        System.Console.WriteLine("Time: " + timeVal) ' date + time
        System.Console.WriteLine("Time: " + timeVal.ToLocalTime)
        System.Console.WriteLine("Time: " + timeVal.ToShortTimeString)
        System.Console.WriteLine("Time: " + timeVal.TimeOfDay.ToString)
        System.Console.WriteLine("Time: " + timeVal.ToString("HH:mm:ss"))
        System.Console.WriteLine(dateVal.ToLongDateString) ' weekday

        ' time measuring
        Dim stopW As New System.Diagnostics.Stopwatch
        stopW.Start()
        Dim counter
start:
        Math.Sqrt(Math.Sin(counter) * Math.Cos(counter))
        counter += 1
        If counter < 20000000.0 Then GoTo start
        stopW.Stop()
        System.Console.WriteLine("elapsed milliseconds=" & stopW.ElapsedMilliseconds)
```

Value types

```
dateVal = Today.AddDays(40)
System.Console.WriteLine("future date: " & dateVal)

Dim duration As System.TimeSpan = Today.AddDays(20).AddHours(10) -
                                  Today.AddDays(14).AddHours(3)

System.Console.WriteLine("difference: " & duration.Days & " days " &
                                          duration.Hours & " hours")
```

Value types

```
    '-----------------------------
    ' enum
    '-----------------------------
    ' Enum MyEnum : ValueOne : ValueTwo : End Enum
    ' error, enums can't appear within a method body

    '-----------------------------
    ' struct
    '-----------------------------
    ' Structure MyStruct : Dim ItemOne : Dim ItemTwo : End Structure
    ' error, structs can't appear within a method body
End Sub ' !

Enum ColorType : RED : GREEN = &HFF00 : BLUE : BLACK : WHITE : YELLOW : End Enum

Public Enum CurrencyType As Integer ' only integral types
    EUR = 1
    USD = 110
    GBP = 80
    JPY = 11000
    CHF
    AUD
    [TRY]
    BRL
    CAD
    HKD = 850
    DKK
    KWD = 33
    THB
    ILS
    NOK
End Enum
```

Value types

```
Structure CustomerType
    Dim customerCompany As String
    Dim contactPerson As String
    Dim id As UShort
    Dim customerAddress As String
    Dim customerSince As Date
    Dim customerSales As Decimal
    Structure phone
        Dim countryCode As String
        Dim areaCode As String
        Dim number As String
        Dim directDial As String
    End Structure
End Structure
```

Value types

```
Sub enumStructTypes()
        '-----------------------------
        ' iterate through enumeration
        '-----------------------------
        ' elements of the array are sorted by the binary values of the enumeration constants
        Dim enumItems As System.Array
        enumItems = System.Enum.GetNames(GetType(CurrencyType)) ' GetNames type string
        For Each item As String In enumItems
            System.Console.WriteLine("enum id: " + item)
        Next

        Dim enumValues As System.Array
        ' or (GetType → Type Currency)
        System.Console.WriteLine(GetType(CurrencyType)) ' → Intro.BasicDatatypes+Currency
        Dim enumIntValues() As Integer = ' conversion to int array
            CType(System.Enum.GetValues(GetType(CurrencyType)), Integer())
        enumValues = System.Enum.GetValues(GetType(CurrencyType))
        For Each item As Integer In enumValues
            System.Console.WriteLine("enum value: " & item)
        Next
```

Value types

```
'-----------------------------
' declare variable of
' enum type
'-----------------------------
Dim color As ColorType
color = ColorType.GREEN
System.Console.WriteLine(color)
System.Console.WriteLine(color.ToString)

' enum value formatting
System.Console.WriteLine(System.Enum.Format(GetType(ColorType), color, "d"))
' or
System.Console.WriteLine([Enum].Format(GetType(ColorType), color, "x"))

Dim currency As CurrencyType
currency = CurrencyType.HKD
Dim invoice As Double = 15980.88
System.Console.WriteLine("invoice total in eur={0:#,##0.00 EUR}", _
                         invoice / currency * 100)
```

Value types

```
        '-----------------------------
        ' declare variable of
        ' structure type
        '-----------------------------
        Dim customer As CustomerType
        customer.customerSales = 380500.45
        customer.id = 720
        Dim customerPhone As CustomerType.phone
        customerPhone.areaCode = "(214)"
    End Sub
End Module
```

Value types, variant types

```
Sub variantTypes()
        '----------------------------
        ' defining mixed types
        '----------------------------
        Dim v1, v2, v3 As Integer, v4, v5 As Single, v6
        ' explicit initalization not allowed when declaring several variables

        '----------------------------
        ' determine type of variant
        '----------------------------
        Debug.Print("v1 is of type " &
                    Global.Microsoft.VisualBasic.VarType(v1)) ' 3
        Debug.Print("v1 is of type " &
                    Global.Microsoft.VisualBasic.VarType(v1).ToString) ' Integer
        Debug.Print("v2 is of type " + VarType(v2).ToString) ' Integer
        Debug.Print("v3 is of type " + VarType(v3).ToString) ' Integer
        Debug.Print("v4 is of type " + VarType(v4).ToString) ' Single
        Debug.Print("v5 is of type " + VarType(v5).ToString) ' Single
        Debug.Print("v6 is of type " + VarType(v6).ToString) ' Object
        ' VB.NET does not support the Variant data type. The Object data type is an universal
        ' datat ype, that can hold data of any other data type
```

Value types, variant types

```
' VarType function returns the type of a variant object defined
' by the VariantType enumeration

' Enum VariantType
'    Empty = 0
'    Null = 1
'    [Short] = 2
'    [Integer] = 3
'    [Single] = 4
'    [Double] = 5
'    Currency = 6
'    [Date] = 7
'    [String] = 8
'    [Object] = 9
'    [Error] = 10
'    [Boolean] = 11
'    [Variant] = 12
'    DataObject = 13
'    [Decimal] = 14
'    [Byte] = 17
'    [Char] = 18
'    [Long] = 20
'    UserDefinedType = 36
'    Array = 8192
' End Enum
```

Value types, variant types

```
' check
If VarType(v6) = Global.Microsoft.VisualBasic.VariantType.Object Then _
    Debug.Print("v1 is an Object")
' alternatively
If VarType(v6) = Global.Microsoft.VisualBasic.Constants.vbObject Then _
    Debug.Print("v1 is an Object")

'----------------------------
' nothing
'----------------------------
Dim var
' check for 'nothing' (literal)
' nothing represents the default value of any data type
' reference type is set to a null reference
' (not associated with any object)
Debug.Print(var Is Nothing) ' True
Debug.Print(IsNothing(var)) ' True
Debug.Print(var = Nothing)  ' True
```

Value types, variant types

```
var = New String("test")
Debug.Print(var Is Nothing) ' False
Debug.Print(IsNothing(var)) ' False
Debug.Print(var = Nothing)  ' False

var = ""
Debug.Print(var Is Nothing) ' False
Debug.Print(IsNothing(var)) ' False
Debug.Print(var = Nothing)  ' True! → avoid '= Nothing' or '<> Nothing'
Debug.Print(var.Equals(Nothing))' False

' check for 'nothing' with value types
Dim i As Integer
i = Nothing ' → value type is set to its default value
Debug.Print(i) ' 0
' Debug.Print(i Is Nothing) ' error, operator can't be applied to value types
Debug.Print(IsNothing(i)) ' False
Debug.Print(i = Nothing) ' True → contradiction, error prone

Dim varT As VariantType ' numeric
Debug.Print(varT) ' 0 ↔ Empty
' Debug.Print(varT Is Nothing) ' error, operator can't be applied
Debug.Print(IsNothing(varT)) ' False
Debug.Print(varT = Nothing)  ' True → contradiction, error prone

' negation
If Not IsNothing(var) Then _
    Debug.Print("var points to an object")
' or
If Not var Is Nothing Then _
    Debug.Print("var points to an object")
' or
If var IsNot Nothing Then _
    Debug.Print("var points to an object")
```

Value types, variant types

```
'----------------------------
' nullable types
'----------------------------
' • declaring a variable as nullable expands the applicable methods
'   with hasValue and Value to indicate whether a value has been assigned
' • underlying type must be a value type
' • 'is' operator can be used
' • assigning nothing to a nullable variable sets the value to null
' • nullable type is constructed from the generic Nullable(Of T) structure

Dim size As Integer
size = Nothing
Debug.Print(size) ' 0 (default value)
' If size.hasValue Then Debug.Print("size has been assigned") ' error
' size is not of type nullable
If IsNothing(size) Then _
    Debug.Print("not assigned") ' no, error prone
If size = Nothing Then _
    Debug.Print("not assigned") ' yes
If size <> Nothing Then _
    Debug.Print("assigned") ' no

size = 0
System.Console.WriteLine(size = Nothing) ' true, still default value
```

Value types, variant types

```
size = 10
If IsNothing(size) Then _
    Debug.Print("not assigned") ' no
If size = Nothing Then _
    Debug.Print("not assigned") ' no, <> default value
If size <> Nothing Then _
    Debug.Print("assigned") ' yes

' example
Dim clientGroup As String
Dim newClient As Boolean
newClient = True
If newClient Then clientGroup = ""
If clientGroup = Nothing Then MsgBox("error, group not initalized") ' yes
If IsNothing(clientGroup) Then
    MsgBox("error, group not initalized") ' but clientGroup was assigned
Else
    MsgBox("already initialized")
End If

' nullable (only for value types)
Dim numClients As System.Nullable(Of UShort)
Dim numOrders? As Integer ' shorthand
numOrders = Nothing
If numOrders.HasValue Then
    Debug.Print("size has been assigned")
    Debug.Print(numOrders.Value)
Else
    Debug.Print("error, not assigned")
End If
```

Value types, variant types

```
numOrders = 10
If numOrders.HasValue Then
    Debug.Print("size has been assigned")
    Debug.Print(numOrders.Value)
End If

'----------------------------
' GetType, GetTypeCode,
' VarType TypeName, TypeOf
'----------------------------
' • GetType returns an instance of Type Class
' • TypeOf works only with reference/variant types (option infer off)
' • TypeName returns a string
' • VarType returns a member of the VariantType enumeration

var = 1
Debug.Print(var.GetType.ToString)  ' System.Int32, qualified name
Debug.Print(var.GetTypeCode)       ' 9
Debug.Print(VarType(var))          ' 3
Debug.Print(VarType(var).ToString) ' Integer
Debug.Print(TypeName(var))         ' Integer, shortname
If TypeOf var Is Integer Then Debug.Print("var is type of integer") ' yes
```

Value types, variant types

```
        var = "test"
        Debug.Print(var.GetType.ToString)  ' System.String
        Debug.Print(var.GetTypeCode)       ' 18
        Debug.Print(VarType(var))          ' 8
        Debug.Print(VarType(var).ToString) ' String
        Debug.Print(TypeName(var))         ' String
        If TypeOf var Is String Then Debug.Print("var is type of string")

        var = Nothing
        ' Debug.Print(var.GetType.ToString) ' error
        ' Debug.Print(var.GetTypeCode)      ' error
        Debug.Print(VarType(var))          ' 9
        Debug.Print(VarType(var).ToString) ' Object
        Debug.Print(TypeName(var))         ' Nothing
        If TypeOf var Is Object Then
            Debug.Print("var is type of object") ' yes
        Else
            Debug.Print("var is nothing")
        End If
        ' if objectexpression is null, then TypeOf...Is returns False,
        ' and ...IsNot returns True
    End Sub
End Module
```

Optionen

```
Option Explicit On ' forces explicit declaration of all variables

Option Strict Off ' forces declaration of datatype → 'as' is required
' speeds up the execution of code, compiled code might have to convert back and
' forth between Object and other data types, which reduces performance, implicit
' data type conversions are disallowed (only 'widening' conversions)

Option Infer On ' declare local variable without explicitly stating a data type,
' compiler infers the data type of a variable from the type of its initialization
' expression: dim i = 1 → i is type of integer

Option Compare Binary ' case sensitive
' Option Compare Text ' not case sensitive
```

Type Conversion

- Parse bzw. TryParse-Methoden
  - Parse throws Exception
    - Prüfausdruck muss in einem Try/Catch-Statement eingebettet sein
      Schema:

```
Try
    …
    .Parse(Expression)
Catch Exception
    Exception Handling
End Try
```

  - TryParse returns Boolean
    - Rückgabewert wird auf True/False geprüft
    - zusätzliche Definition einer Variablen, die das geparste Ergebnis speichert
- Methoden der Convert-Klasse
- automatische Typumwandlung

Type Conversion

```
Module Conversions
    Sub conversion()
        '============================
        ' string → int
        '============================

        '----------------------------
        ' Parse/TryParse
        '----------------------------

        ' Parse throws exception on failing
        ' TryParse returns a bool

        Dim i? As Integer = Nothing  ' Nothing for indicating a parser error

start: ' Parse
        Try ' exception handling
            Select Case CInt(
                Global.Microsoft.VisualBasic.Interaction.InputBox("select 1..6"))
                Case 1
                    i = Integer.Parse("true") 'error
                Case 2
                    i = Integer.Parse("123")
                Case 3
                    i = Integer.Parse("-000123")
                Case 4
                    i = Integer.Parse("1.23") 'error
                Case 5
                    i = Integer.Parse("000123ABC") 'error
                Case 6
                    Exit Sub
            End Select
```

Type Conversion

```
        Catch e As System.Exception
            System.Console.WriteLine("error, can't parse expression")
        End Try

        ' check for nothing
        System.Console.WriteLine("i=" + If(i.HasValue, i.ToString(), "null value"))
        ' Or
        System.Console.WriteLine("i=" + If(i IsNot Nothing, i.ToString(), "null value"))
        ' Or
        System.Console.WriteLine("i=" +
            If(Global.Microsoft.VisualBasic.Information.IsNothing(i),
               "null value",
               i.ToString()))

        If i.HasValue Then
            Microsoft.VisualBasic.MsgBox("i=" & i)
        End If

        i = Nothing
        GoTo start
    End Sub
End Module
```

Type Conversion

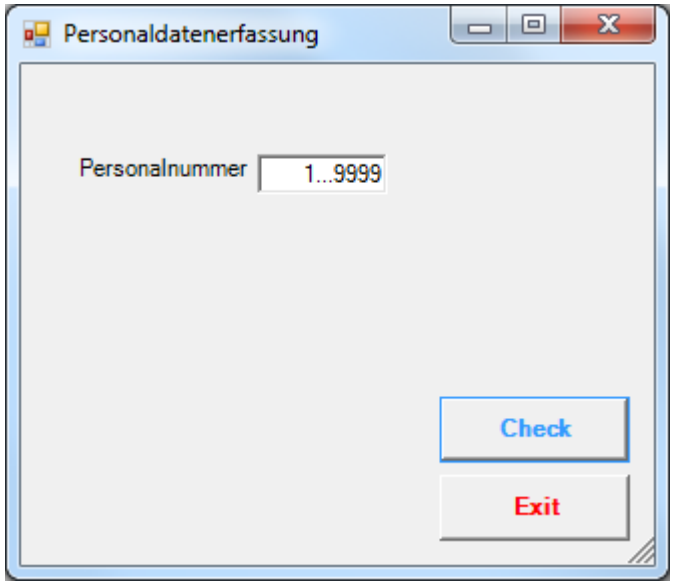
```
Module Conversions
    Sub conversion()
        ' TryParse
        Dim i? As Integer = Nothing ' nullable as indicator
        Dim outInt As Integer ' storage can't be nullable

        If Integer.TryParse("true", outInt) Then ' throws no exception, check
                                                 ' for true or false
            i = outInt
            System.Console.WriteLine("successfully parsed, i=" & i)
        Else
            System.Console.WriteLine("error, can't parse expression")
        End If
```

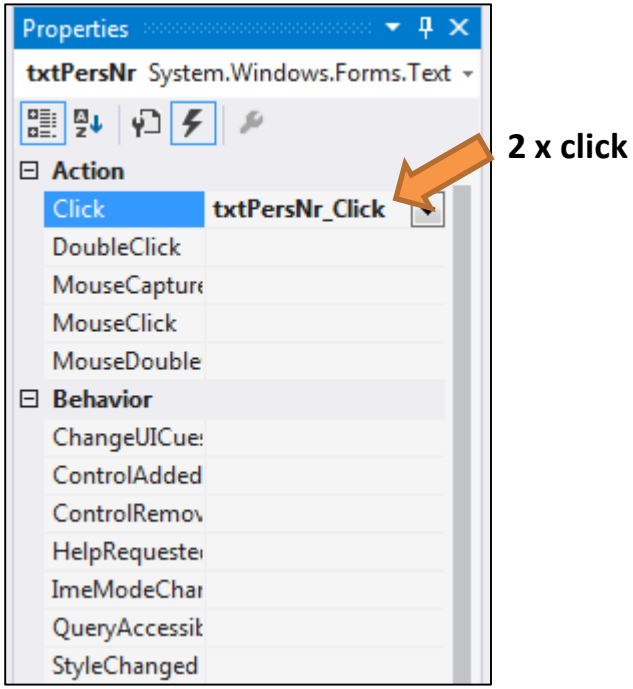
Type Conversion

```
        i = Nothing ' reset

        If Integer.TryParse("  123  ", outInt) Then
            i = outInt
            System.Console.WriteLine("successfully parsed, i=" & i)
        Else
            System.Console.WriteLine("error, can't parse expression")
        End If

        Dim v As Nullable(Of Boolean) ' alternative definition
        Dim outBool As Boolean

        If Boolean.TryParse("  true  ", outBool) Then
            v = outBool
            System.Console.WriteLine("successfully parsed, v=" & v)
        Else
            System.Console.WriteLine("error, can't parse expression")
        End If
    End Sub
End Module
```

Type Conversion

- Beispielanwendung Formular
  - Prüfroutine Textfeld





  - Properties
    - Properties
      - Text, TextAlign, (Name), Font Bold, ForeColor 255;0;0
    - Events
      - Click
        - Inhalt löschen nach erstmaligem Klick
      - Keypress
        - mittels Enter-Taste Check-Routine aufrufen

Type Conversion

```
Public Class frmPersonalDaten

    Private Sub txtPersNr_Click(sender As Object, e As EventArgs) Handles txtPersNr.Click
        Static changed As Boolean
        If Not changed Then txtPersNr.Clear()
        changed = True
    End Sub

    Private Sub cmdCheck_Click(sender As Object, e As EventArgs) Handles cmdCheck.Click
        Dim persNr As Integer
        If String.IsNullOrEmpty(txtPersNr.Text) Then
            MsgBox("Personalnummer eintragen")
        ElseIf Not Integer.TryParse(txtPersNr.Text, persNr) Then
            MsgBox("Keine gültige Personalnummer")
            txtPersNr.Clear()
        ElseIf persNr < 1 Or persNr > 9999 Then
            MsgBox("Personalnummer zu klein/gross")
        Else
            MsgBox("accepted")
        End If
        txtPersNr.Focus()
    End Sub
```

Type Conversion

```
    Private Sub txtPersNr_KeyPress(
        sender As Object,
        e As System.Windows.Forms.KeyPressEventArgs) Handles txtPersNr.KeyPress
        If e.KeyChar = Microsoft.VisualBasic.ChrW(System.Windows.Forms.Keys.Return) Then
            cmdCheck_Click(Nothing, Nothing)
            e.Handled = True
        End If
    End Sub

    Private Sub cmdExit_Click(sender As Object, e As EventArgs) Handles cmdExit.Click
        persForm.Close()
    End Sub
End Class

Module Personalverwaltung
    Public persForm As New Global.Intro.frmPersonalDaten
    Sub main()
        persForm.ShowDialog()
    End Sub
End Module
```

Type Conversion

- Schema Try/Catch/End Try

```
Try
Catch more special error
Catch less special error
...
Catch general error
End Try
```

```
Module Parsing
    Sub fileReadWrite()
        '----------------------------
        ' Example Parse file input
        '----------------------------
        Dim filePath As String = "C:\Users\" + System.Environment.UserName +
                                 "\Desktop\order_dispatch.txt"

        ' write file
        Const length As Integer = 100
        Try
            Dim fs As New System.IO.FileStream(filePath, System.IO.FileMode.Create)
            ' filestream for writing bytes or block of bytes
            Dim sw As New System.IO.StreamWriter(fs)
            ' StreamWriter implements abstract class TextWriter

            sw.WriteLine("{0,7} {1,8}", "OrderID", "Quantity")
            Dim randomQuant As String = "0123456789AB", quantity
            Dim rnd As New System.Random() ' not in loop!
```

Type Conversion

```
        For k As Integer = 1 To length
            quantity = ""
            For m As Byte = 0 To 5 ' generate random quantity
                quantity += randomQuant.ToCharArray()(
                                rnd.Next(randomQuant.Length))
                                ' length: 12, index: 0..11
            Next
            sw.WriteLine("{0,7:000000} {1,8}", k, quantity)
        Next
        sw.Close()
    Catch e As System.Exception
        System.Console.WriteLine(e.Message)
    End Try

    ' read file
    Dim s As String, i, j As Integer
    Try
        Dim fr As New System.IO.FileStream(filePath, System.IO.FileMode.Open)
        Dim sr As New System.IO.StreamReader(fr)
```

Type Conversion

```
s = sr.ReadLine
While s IsNot Nothing
    System.Console.WriteLine("s=" + s)
    Try

        ' tokenize line, default delimiter blank
        ' split-method: _XX___XXX_X → _, XX_, _, _, XXX_, X
        i = Integer.Parse(s.Split()(4)) ' jump to catch if parse or split fails
        System.Console.WriteLine("i=" & i)
    Catch e As Exception
        System.Console.WriteLine("splitting or parsing error")
        System.Console.WriteLine(e.Message)
    End Try
```

Type Conversion

```
            ' Or
            Try
                s = s.Split()(4) ' first check split for preparing tryparse
            Catch
                System.Console.WriteLine("splitting error")
            End Try
            If Integer.TryParse(s, j) Then
                System.Console.WriteLine("j=" & j)
            Else
                System.Console.WriteLine("parsing error")
            End If
            s = sr.ReadLine
        End While
        sr.Close()
    Catch e As System.IO.DirectoryNotFoundException
    Catch e As System.IO.FileNotFoundException
    Catch e As Exception
        System.Console.WriteLine(e.Message)
    End Try
```

Type Conversion

```
        ' Or
        For Each item As String In System.IO.File.ReadAllLines(filePath)
            System.Console.WriteLine("item=" + item)
            ' the first dataset contains only 2 delimiters (blanks)
            If item.Split().Length > 2 AndAlso Integer.TryParse(item.Split()(4), j) Then
            ' lazy evaluation, else splitting error
                System.Console.WriteLine(j)
            End If
        Next
        ' automatic close of ReadAllLines-method
    End Sub
End Module
```

Type Conversion

```
Sub main()
        '----------------------------
        ' Convert Class
        '----------------------------
        Dim str As String = "123", i As Integer
        i = System.Convert.ToInt32(str)

        Dim nBase As Integer ' Convert with numerical base
        nBase = 2
        str = "1001010010111011"
        i = System.Convert.ToInt32(str, nBase)
        System.Console.WriteLine("1001010010111011(2)=" & i)

        nBase = 8
        str = "77770000"
        i = System.Convert.ToInt32(str, nBase)
        System.Console.WriteLine("77770000(8)=" & i)

        nBase = 16
        str = "FFFFFF"
        i = System.Convert.ToInt32(str, nBase)
        System.Console.WriteLine("FFFFFF(16)=" & i)
```

Type Conversion

```
'============================
' int → string
'============================
i = 1
str = i ' ok
str = i.ToString()
System.Console.WriteLine("str=" + str)

i = 123
str = i
str = System.Convert.ToString(i)
cSystem.Console.WriteLine("str=" + str)

'============================
' single → string
'============================
Dim s! = 1.223344F
str = s
System.Console.WriteLine(str) '= 1,223344
```

Type Conversion

```
'============================
' char → int
'============================
Dim c As Char = "A"c
System.Console.WriteLine(Global.Microsoft.VisualBasic.Asc(c)) '= 65
System.Console.WriteLine(System.Convert.ToInt32(c)) '= 65

' int value of char digits
i = Char.GetNumericValue("7") ' returns a double
System.Console.WriteLine(i) '= 7

i = Char.GetNumericValue("9876543210", 3) ' zero-based index
System.Console.WriteLine(i) '= 6

'============================
' int → char
'============================
i = 1
' c = 1 ' error
c = Global.Microsoft.VisualBasic.ChrW(i)
Dim space As Char = ChrW(32)
System.Console.WriteLine("blank" + space + "blank" + space + "blank")

c = System.Convert.ToChar(1)
```

Type Conversion

```
        '============================
        ' bool → int
        '============================
        Dim b As Boolean = False
        i = b ' ok
        System.Console.WriteLine(i) '= 0
        System.Console.WriteLine(System.Convert.ToInt32(False)) '= 0
        System.Console.WriteLine(System.Convert.ToInt32(b))     '= 0
        b = True
        System.Console.WriteLine(System.Convert.ToInt32(True)) '= 1
        System.Console.WriteLine(System.Convert.ToInt32(b))     '= 1

        '============================
        ' int → bool
        '============================
        i = 0
        b = i ' ok
        System.Console.WriteLine("b=" & b) '= False
        System.Console.WriteLine("int to bool,  0=" & System.Convert.ToBoolean(0))  '= False
        i = 2
        System.Console.WriteLine("int to bool,  2=" & System.Convert.ToBoolean(2))  '= True
        i = -1
        System.Console.WriteLine("int to bool, -1=" & System.Convert.ToBoolean(-1)) '= True
    End Sub
End Module
```

Übung - Datenkonversion

a) Konvertieren Sie die folgenden Werte:

```
Dim s1 As String = "2835"            ' nach integer
Dim s2 As String = "-12534.54"       ' nach single
Dim i1 As Integer = 13303790         ' in hexadezimaler Schreibweise, Großbuchstaben
Dim i2 As Integer = 50400            ' in binärer Schreibweise
Dim s3 As String = "BA0DE1"          ' nach integer
Dim s4 As String = "100101011001101" ' nach integer
```

Lösung - Datenkonversion

a)

```
Dim s1 As String = "2835"             ' nach integer
System.Console.WriteLine(System.Convert.ToInt32(s1))

Dim s2 As String = "-12534.54"        ' nach single
System.Console.WriteLine(System.Convert.ToSingle(s2))

Dim i1 As Integer = 13303790          ' in hexadezimaler Schreibweise, Großbuchstaben
System.Console.WriteLine(System.Convert.ToString(i1, 16).ToUpper())

Dim i2 As Integer = 50400             ' in binärer Schreibweise
System.Console.WriteLine(System.Convert.ToString(i1, 2))

Dim s3 As String = "BA0DE1"           ' nach integer
System.Console.WriteLine(System.Convert.ToInt32(s3, 16))

Dim s4 As String = "100101011001101" ' nach integer
System.Console.WriteLine(System.Convert.ToInt32(s4, 2))
```

Operatoren

```
Module LazyEvaluation
    Sub main()
        Dim i As Integer = 1

        If i < 1 And check(1) Then ' 0 → runtime error
            System.Console.WriteLine("true case")
        Else
            System.Console.WriteLine("false case")
        End If

        If i < 1 AndAlso check(0) Then
            System.Console.WriteLine("true case")
        Else
            System.Console.WriteLine("false case")
        End If
    End Sub

    Function check(Optional i As Integer = 1) As Boolean
        Dim j As Integer = 10 / i
        If j > 5 Then Return True Else Return False
    End Function
End Module
```

Operatoren - Übung

d) Gegeben sei ein Gatterschaltkreis mit vier Eingängen a, b, c, d und der Schaltfunktion q = $(\neg a \wedge \neg b \wedge \neg c \wedge \neg d) \vee (\neg a \wedge b \wedge \neg c \wedge \neg d) \vee (\neg a \wedge \neg b \wedge c \wedge \neg d)$.
Setzen Sie q in einen äquivalenten Ausdruck um.

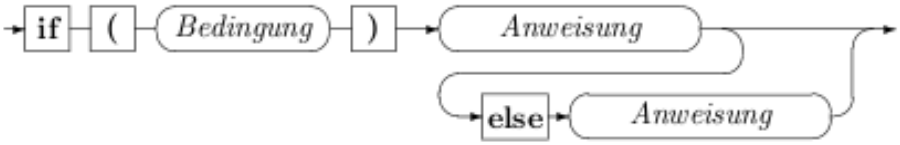
e) Berechnen Sie die folgenden Ausdrücke:

```
int i=10;
i<<=2;
boolean b=i!=i++||i==41;
i=i+++10+i;
```

f) Berechnen Sie das Kugelvolumen K nach der Formel
K = 4/3 x Pi x Radius^3

Kontrollstrukturen

- Bedingungen
  - if

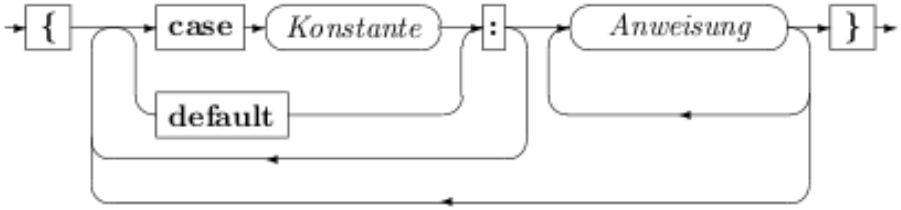


    if Bedingung then Statement

    if Bedingung then Statement1 else Statement2

  - select/case

```
select case selector
{
  case constant1
     anweisung1
  case constant2
     anweisung2
case constant3:
     anweisung3
case default
     anweisung4
end select
```



  - Bedingungsoperator

    x = if(a > b , a , b)

Kontrollstrukturen

<u>if else</u>

```
class ControlFlow
    static void conditions()
        int wert = 17;
        if (wert < 10)
            System.out.println("Wert ist kleiner 10!");
        } else if ((wert >= 10) && (wert < 20))
            System.out.println("Wert ist groesser oder"
                    + " gleich 10 und kleiner 20!");
        } else if ((wert >= 20) && (wert < 30))
            System.out.println("Wert ist groesser oder"
                    + " gleich 20 und kleiner 30!");
        } else
            System.out.println("Wert ist groesser 30!");
        }
    }
}
```

Kontrollstrukturen


```
Scanner In = new Scanner(System.in);
System.out.println("Projektstatus eingeben:");
byte PS = In.nextByte();

if (!(PS >= 0 && PS <= 100))
{
    System.out.println("Error!");
    Runtime.getRuntime().exit(0);
}

if (PS >= 0 && PS <= 25)
    System.out.println("Anfangsphase");
else if (PS >= 26 && PS <= 70)
{
    System.out.println("Mittlere Phase");
    System.out.println("Bericht? (j/n)");
    Scanner ScIn = new Scanner(System.in);
    String c = ScIn.next();
    if (c.compareTo("j") == 0) System.out.println("Bericht");
}
```

Kontrollstrukturen

```
int s=0;
for (int i = 0; i < 100; i++) {
    s+=i;
}

double kapital = 0.00;
int jahr = 0;
while (jahr<10) {
    kapital = kapital * 1.06 + 2400;
    jahr = jahr + 1;
    System.out.print(jahr);
    System.out.print(" - ");
    System.out.println(kapital);
}
```

Kontrollstrukturen


```java
import java.util.Scanner;

String pwd = "Himitsu", pInput;
final byte maxTries = 5;
byte nTries = 0;

Scanner ScIn = new Scanner(System.in);
do {
    System.out.println("Passwort eingeben:");
    pInput = ScIn.next();
    nTries++;
    if (pInput.compareTo(pwd) == 0) {
        System.out.println("Zugang gewährt!");
        nTries = maxTries;
    } else if (nTries == maxTries) {
        System.out.println("GESPERRT!!");
    } else {
        System.out.println("Falsches Passwort!");
    }
} while (nTries < maxTries);
```

For Each

```
Module ForEach
    Sub main()
        Dim list As New List(Of String)

        For i As Integer = 1 To 10
            list.Add(Console.ReadLine)
        Next

        For Each item As String In list
            System.Console.WriteLine("item=" + item)
        Next

        Dim singleList As New List(Of Single)
        Dim rnd As New Random

        For i As Integer = 1 To 100
            singleList.Add(rnd.NextDouble * 10)
        Next
```

For Each

```
        MsgBox("unsortiert")
        For Each item As Single In singleList
            System.Console.WriteLine("item=" & item)
        Next

        singleList.Sort()

        MsgBox("sortiert")
        For Each item As Single In singleList
            System.Console.WriteLine("item=" & item)
        Next

        singleList.Reverse()

        MsgBox("sortiert, absteigend")
        For Each item As Single In singleList
            System.Console.WriteLine("item=" & item)
        Next
    End Sub
End Module
```

Kontrollstrukturen - Übung

a) Es ist eine einfache Grafikroutine zu entwickeln, die prüft, ob ein vorzeichenbehaftetes Zahlenpaar (XY-Koordinate) im I, II, III oder IV Quadranten eines Koordinatensystems liegt.

   Die Routine soll folgende Ausgabe leisten:
   für +x, +y : Ausgabe "Q1(++)"
   für +x, -y : Ausgabe "Q4(+-)"
   für -x, +y : Ausgabe "Q2(-+)"
   für -x, -y : Ausgabe "Q3(--)"

   Prüfen Sie die Abfragelogik innerhalb einer Schleife mit zufälligen Werten. Verwendend Sie dazu die Methode <u>random</u> der *Math*-Klasse.

   Lösungshinweis: Erstellen Sie im Bedarfsfall zunächst ein Flussdiagramm (Programmablaufplan) oder Nassi-Shneiderman-Diagramm, welches die Zerlegung des Gesamtproblems in Teilprobleme visualisiert.

Kontrollstrukturen - Übung

b) Die Simulation soll nun grafisch in einem Diagramm unter Verwendung der JFrame-Klasse dargestellt werden. Integrieren Sie dazu das Programm aus Teil a) in folgendes Codefragment:

Java.txt

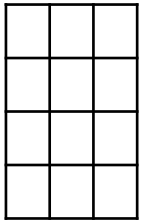
```
a)  import java.awt.Color;
b)  import java.awt.Graphics;
c)  import javax.swing.JFrame;

d)  public class Frame extends JFrame {
e)    public static void main(String[] args) { new Frame(); }
f)    Frame() {
g)      setSize(400, 400);
h)      setVisible(true);
i)      setResizable(false);
j)      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
k)    }
l)    public void paint(Graphics g) {
m)      int x, y;
n)      g.drawLine(0, 200, 400, 200);
o)      g.drawLine(200, 0, 200, 400);

p)      /*
q)        Code ergänzen
r)      */

s)      setIgnoreRepaint(true);
t)    }
u)  }
```

Felder (Arrays)

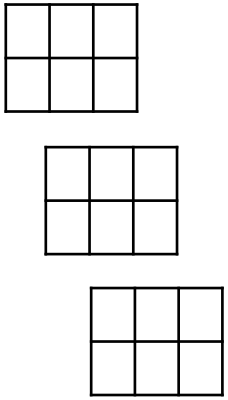
- Datenstruktur für die Speicherung typgleicher Daten
- Def.: Dim FeldVariable(n) As Datentyp, n ≥ 0
- Indexierung von 0 bis n ⇨ Speicherung von n + 1 Elementen
- Eindimensionale, mehrdimensionale Feldstruktur
  - Eindimensional
    - Dim FeldVariable(n) As Datentyp
  - Mehrdimensional, "rechteckiges" Schema (≙ Tabelle)
    - Dim FeldVariable(n, m) As Datentyp
  - "verzweigtes" Schema, Array aus Arrays mit unterschiedlicher Länge, sog. jagged Arrays
    - Dim FeldVariable(n)( ) As Datentyp
- Zugriff
  - FeldVariable(n) = Wert
  - Schleife
    - For Next
    - For Each
- Ermittlung der oberen Feldgrenze
  - Feldvariable.GetUpperBound(Rang), Rang ≙ Dimension, Rang ≥ 0
  - Global.Microsoft.VisualBasic.UBound(FeldVariable, Dimension), Rang ≥ 1

Felder

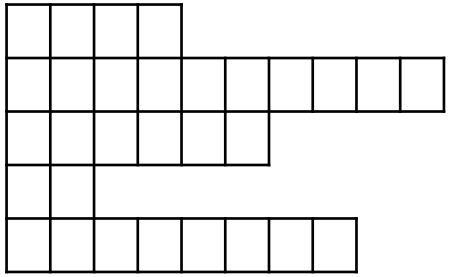
- 2-dimensional: n x m (rechteckig)
- 3-dimensional: r x s x t (quaderförmig)
- verzweigte Arrays (Array aus Arrays) mit unterschiedlicher Länge (jagged)



4 x 3



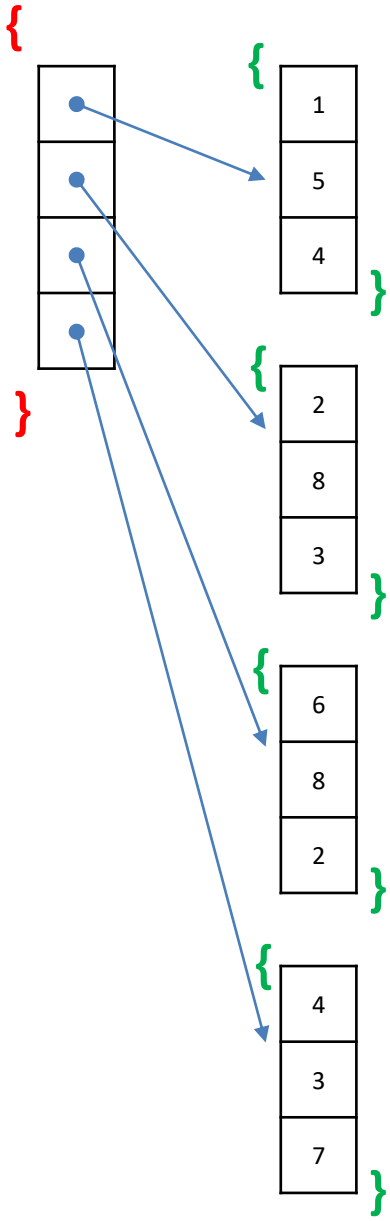
3 x 2 x 3



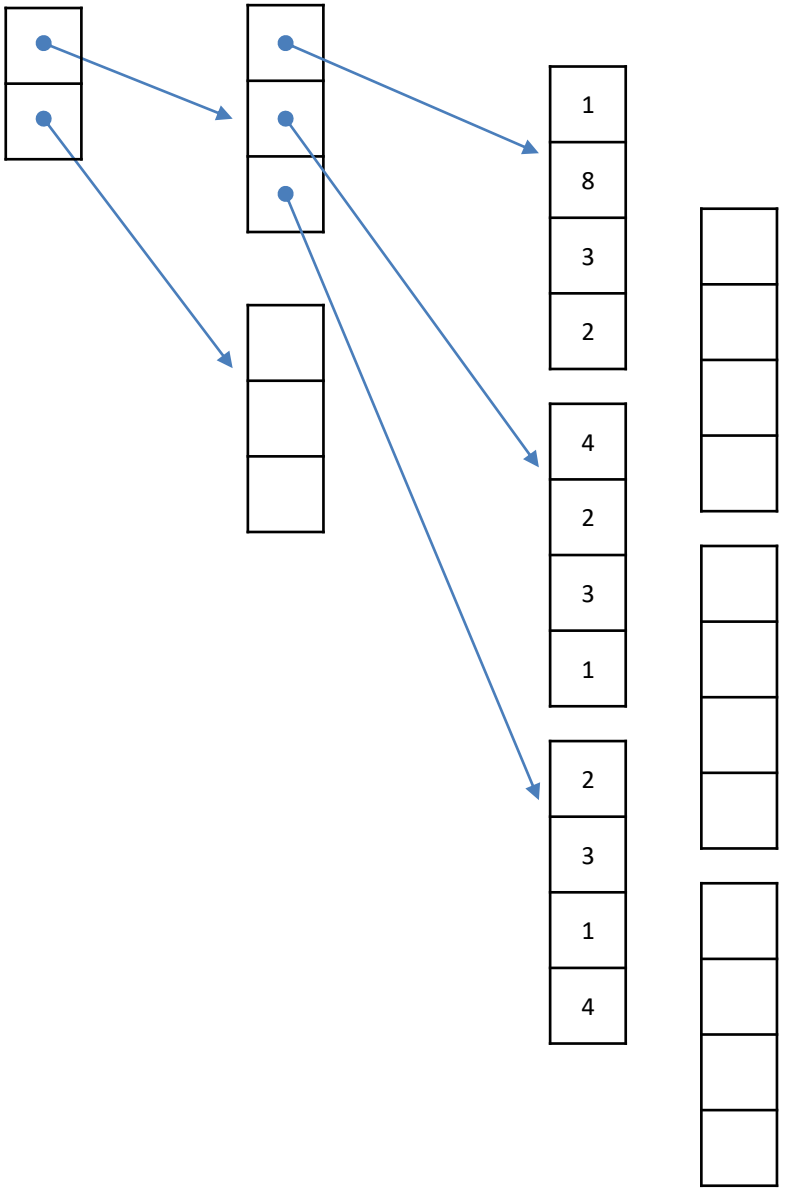
jagged array

Felder - 4 x 3 Matrix

```
Dim b(,) = {{1, 5, 4}, {2, 8, 3}, {6, 8, 2}, {4, 3, 7}}
```



Felder - 2 x 3 x 4 Matrix




```
Dim c(,,) = {
                {
                  {1, 8, 3, 2},
                  {4, 2, 3, 1},
                  {2, 3, 1, 4}
                },
                {
                  {2, 7, 4, 9},
                  {1, 3, 4, 5},
                  {2, 4, 3, 6}
                }
            }
```

Felder

```
Imports System.Console

Module Arrays
    Sub Main()
        Dim doubleArray(100) As Double ' zero-initialized, index from 0 to 100 => 101 values
        Dim dimension = 1
        For index = 0 To Global.Microsoft.VisualBasic.UBound(doubleArray, dimension)
            doubleArray(index) = Global.Microsoft.VisualBasic.Rnd * 10000 ' Rnd: [0..1[
        Next

        ' output value
        For Each item In doubleArray
            WriteLine("value={0}", item)
        Next

        ' output formatted index and value
        For index = 0 To Global.Microsoft.VisualBasic.UBound(doubleArray, dimension)
            WriteLine("{0,4:\#0} value={1:F2}", index, doubleArray(index)) ' \ escape
        Next
        ReadKey()

        ' write file
        Dim file As String = "C:\Users\" + System.Environment.UserName + "\Desktop\data.txt" ' vb.net doesn't support inline
                                                                                              ' control characters
        Using writer As System.IO.StreamWriter = New System.IO.StreamWriter(file)
            For index = 0 To Global.Microsoft.VisualBasic.UBound(doubleArray, dimension)
                writer.WriteLine("{0,4:\#0} value={1:F2}", index, doubleArray(index)) ' \ escape
            Next
        End Using
        WriteLine("written")
        ReadKey()

        ' read file
        Using reader As System.IO.StreamReader = New System.IO.StreamReader(file)
            Dim line As String
            line = reader.ReadLine
            Do While (Not line Is Nothing)
                WriteLine(line)
                line = reader.ReadLine
            Loop
        End Using
        WriteLine("done")
        ReadKey()
```

Felder

```
' upperbound, lowerbound
WriteLine("upperbound:" & UBound(doubleArray), 1)
WriteLine("upperbound:" & doubleArray.GetUpperBound(0))
WriteLine("lowerbound:" & LBound(doubleArray), 1)
WriteLine("lowerbound:" & doubleArray.GetLowerBound(0))
ReadKey()

' multidimensional array
Dim intArray(3, 10, 5) As Integer
Randomize() ' init the random generator
For i = 0 To 3
    For j = 0 To 10
        For k = 0 To 5
            intArray(i, j, k) = CInt(Rnd() * 100) + 1 ' random values from 1 to 100
            WriteLine("i:{0} j:{1} k:{2} value:{3}", i, j, k, intArray(i, j, k))
        Next
    Next
Next

' get the dimensions
Dim rank1 = UBound(intArray, 1)
Dim rank2 = UBound(intArray, 2)
Dim rank3 = UBound(intArray, 3)

WriteLine("first dimension: " & rank1)
WriteLine("second dimension: " & rank2)
WriteLine("third dimension: " & rank3)

' initializing
Dim a() As Integer = {1, 2, 3, 4, 5}
' or
a = New Integer() {1, 2, 3, 4, 5}
For Each item In a
    WriteLine(item)
Next

Dim b(,) As Integer = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}, {10, 11, 12}} ' 4 x 3
For i = 0 To UBound(b, 1)
    For j = 0 To UBound(b, 2)
        WriteLine(i & " " & j & " " & b(i, j))
    Next
Next
```

Felder

```
Dim c(,,) As Byte = {{{1, 8, 3, 2}, {4, 2, 3, 1}, {2, 3, 1, 4}}, {{2, 7, 4, 9}, {1, 3, 4, 5}, {2, 4, 3, 6}}} ' 2 x 3 x 4
Dim d() As Object = {"abc", 1, 1.0, "A"c, #11/3/2010#}

' nonrectangular, jagged arrays
Dim jagged(10)() As Byte ' declaring top-level upper bound
jagged(0) = New Byte(2)   {}
jagged(1) = New Byte(10)  {}
jagged(2) = New Byte(30)  {}
jagged(10) = New Byte(50) {}

' or, top-level index in the new clause
Dim jaggeda()() As Byte = New Byte(20)() {}
jaggeda(0) = New Byte(20)  {}
jaggeda(1) = New Byte(10)  {}
jaggeda(2) = New Byte(30)  {}
jaggeda(20) = New Byte(50) {}

' with initalization
Dim jaggedb(5)() As Integer
jaggedb(0) = New Integer(3) {1, 2, 3, 4} ' bzw. New Integer() {1, 2, 3, 4}
jaggedb(1) = New Integer(2) {1, 2, 3}
jaggedb(5) = New Integer(4) {1, 2, 3, 4, 5}

' or
Dim jaggedc()() = New Byte(3)() {New Byte() {1, 2}, New Byte() {0, 1, 2}, New Byte() {3, 4, 5, 6}, New Byte() {0}}

Dim jaggedd()() As Short = New Short()() {New Short() {2S, 3S}, New Short() {1S, 4S, 5S, 3S}, New Short() {1S},
                                          New Short() {2S, 5S, 8S}}
For i = 0 To UBound(jaggedd, 1)
    For j = 0 To UBound(jaggedd(i), 1)
        Console.Write(jaggedd(i)(j))
    Next
    Console.WriteLine()
Next

' or double brace initialization
Dim jaggede()() As Byte = {New Byte() {2, 1}, New Byte() {3, 0}}
```

Felder

```
        ' Array Class
        ' copy
        Dim measurementValues(100) As Double
        For i = 0 To 100
            measurementValues(i) = Rnd()
        Next
        Dim copy(100) As Double
        System.Array.Copy(measurementValues, copy, measurementValues.Length)
        For Each item In copy
            WriteLine(item)
        Next
        Console.WriteLine(Microsoft.VisualBasic.StrDup(20, "-"))
        Console.ReadKey()

        ' sort copy
        Array.Sort(copy)
        For Each item In copy
            WriteLine(item)
        Next
        Console.WriteLine(Microsoft.VisualBasic.StrDup(20, "-"))
        Console.ReadKey()

        ' reverse
        Array.Reverse(copy)
        For Each item In copy
            WriteLine(item)
        Next
        Console.WriteLine(Microsoft.VisualBasic.StrDup(20, "-"))
        Console.ReadKey()

        ' Array ForEach
        Dim iArray() As Integer = {2, 3, 4}
        Dim action As New System.Action(Of Integer)(AddressOf ShowSquares)
        Global.System.Array.ForEach(iArray, action)
    End Sub

    Sub ShowSquares(val As Integer)
        Console.WriteLine("{0:d} squared = {1:d}", val, val * val)
    End Sub
End Module
```

Felder

```
' Im folgenden Beispiel wird eine Arrayvariable deklariert,
' die ein Array von Arrays enthalten soll, dessen Elemente
' zum Double - Datentyp(Visual Basic) gehören.
' Jedes Element des Arrays sales ist selbst ein Array,
' das einen Monat darstellt.
' Jedes Monatsarray enthält Werte für jeden Tag dieses Monats.

Dim sales()() As Double = New Double(11)() {}
Dim month As Integer
Dim days As Integer
For month = 0 To 11
days = DateTime.DaysInMonth(Year(Now), month + 1)
sales(month) = New Double(days - 1) {}
Next month
```

Felder

```
Sub arrays()
    Dim varField(5)
    varField(0) = 1
    System.Console.WriteLine(varField.GetLowerBound(0)) '0
    System.Console.WriteLine(varField.GetUpperBound(0)) '5
    System.Console.WriteLine(varField.GetLength(0)) '6

    Dim list(0 To 330) As Integer
    Dim field(434, 223, 95) As Integer
    Dim l As List(Of Integer) = New List(Of Integer)
    ReDim list(10)

    System.Console.WriteLine(list.Length)
    System.Console.WriteLine(list.Count)

    l.Clear()
    Dim r As New Random()
    Dim rand%
    Do
        rand = r.Next(15) + 1
        If Not l.Contains(rand) Then l.Add(rand)
    Loop Until l.Count = 10

    l.Sort()
    For Each item As String In l
        System.Console.WriteLine(item)
    Next
```

Felder

```
    Dim i = 0
    System.Console.WriteLine("")
    Do
        rand = r.Next(15) + 1
        If Not list.Contains(rand) Then
            list(i) = rand
            i += 1
        End If
    Loop Until i = 10

    Array.Sort(list)
    For Each item As String In list
        System.Console.WriteLine(item)
    Next
End Sub
```

Funktionen - Übung

a) Implementieren Sie die Funktion

`float getMwstNetto( float netto, float mSatz )`,

die als Argumente den Nettobetrag und den aktuell gültigen Mwst-Satz bekommt.

Berechnet werden soll die Mwst. auf den Nettobetrag.

b) Implementieren Sie die Funktion

`float getMwstBrutto( float brutto, float mSatz )`,

die als Argument den Bruttobetrag und den aktuell gültigen Mwst-Satz bekommt.

Berechnet werden soll die in einem Bruttobetrag enthaltene Mwst.

In der Funktion soll auf 2 NK-Stellen gerundet werden (Math.round)

Funktionen - Übung

c) Berechnen Sie das Skalarprodukt zweier 3-dimensionaler Vektoren.

Das Produkt berechnet sich aus |v1| x |v2| x cos(alpha).

Der Betrag eines Vektors berechnet sich mit

|v| = Wurzel(vx^2 + vy^2 + vz^2).

Verwenden Sie für die Vektordarstellung ein einfaches float-Array.

Beachten Sie, das der Winkel im Bogenmaß angegeben werden muss.

(Wurzel, Kosinus, Potenz und Bogenmaß findet sich in Math.*)

d) Berechnen Sie den Gesamtwiderstand beliebig vieler

parallelgeschalteter Widerstände

Der Gesamtwiderstand berechnet sich mit 1/Rg = 1/R1 + 1/R2 + ...

1/Rn.

Modul - Aufbau

ModulName

Deklarationsteil

Optionen: `Option Explicit`

Globale Variablen und Konstanten mit Gültigkeit auf Projektebene

```
Public gFirma
Global gOrt
Global Const gPLZ = 33333
```

Variablen und Konstanten mit Gültigkeit auf Modulebene

```
Private mLagerID
Dim mLagerBestandTotal
Const mLagerOrt = "Gütersloh"
```

Prozedur- und Funktionsteil

Globale Prozeduren u. Funktionen

```
Public Sub LagerInfo()
End Sub
```

Prozeduren mit Gültigkeit auf Modulebene

```
Private Sub Disposition()
End Sub
```

Module

```
Sub, function disallowed at file scope
Sub Main()       ' error
End Sub
Sub message()    ' error
End Sub
Function func() ' error
End Function

' every module may have its own main routine,
' to prevent ambiguousness, startup object has to call
' the specific module including the main routine

Module Alpha
    Sub Main()
        MsgBox("Module Alpha")
    End Sub
    Sub Main(i As Integer) ' ok, overloaded
        MsgBox("Module Alpha " + CStr(i))
    End Sub
End Module
```

Module

```
Module Beta
    Sub Main()
        MsgBox("Module Beta")
    End Sub
    Sub message()
        MsgBox("Module Beta")
    End Sub
End Module

' a routine can't contain a module
Module Gamma
    Sub message()
        Module Delta ' error
            MsgBox("Module Gamma")
        End Module
    End Sub
End Module
```

Module

```
' a module can't contain a module
Module Delta
    Module Epsilon ' error
    End Module
End Module

' a module can't contain a namespace
Namespace Peta
    Module Alpha
        Namespace Tera ' error
        End Namespace
    End Module
    Module Beta
    End Module
End Namespace

Module Peta ' error, name conflict
End Module
```

Module

```
' namesspaces can be nested and may contain some modules
Namespace Tera
    Namespace Giga
        Namespace Mega
            Module Alpha
                Sub message()
                    MsgBox("Module Alpha")
                End Sub
            End Module
            Module Beta
                Sub message()
                    MsgBox("Module Beta")
                End Sub
            End Module
        End Namespace
    End Namespace
End Namespace
```

Module - Scope and visibility

```
Dim g_ivar%    ' error
Public p_svar! ' error

Module ScopeVisibility1 ' a module is static and not instantiatable
    Dim iVar% ' default private
    Public sVar!
    Private dVar&
    Static siVar% ' error

    Sub proc() ' default public
        Dim iVar%
        Public sVar!  ' error
        Private dVar& ' error
        Static siVar%
    End Sub

    Private Sub privProc()
    End Sub
End Module
```

Module

```
Module ScopeVisibility2
    Private Sub useGlobalDefinitions()
        iVar = 1 ' error, only visible in module
        ScopeVisibility1.iVar = 1 ' error
        sVar = 1 ' ok, public
        ScopeVisibility1.sVar = 1
        dVar = 1 ' error, private
        proc()   ' ok, default public
        ScopeVisibility1.proc() ' fully qualified
        siVar = 1  ' error, static but private
        privProc() ' error, private
        ScopeVisibility1.privProc() ' error, private
    End Sub
End Module
```

Module - Ambiguities (Mehrdeutigkeiten)

```
Module Module1
    Public gLändercode
    Dim mKundenID
    Dim mFirma
    Dim mAdresse

    Sub init()
        mKundenID = 222
        mFirma = "ABC GmbH"
        mAdresse = "50999 Köln"
    End Sub

    Sub zeigeKundenInfo(typ)
        If typ Then
            Call init()
        Else
            Call Module2.init()
        End If

        Debug.Print("ID =" + mKundenID)
        Debug.Print("Firma  " + mFirma)
        Debug.Print("Adresse = " +
                    mAdresse)
    End Sub
End Module
```

```
Module Module2
    Public gLändercode

    Dim mKundenID
    Dim mFirma
    Dim mAdresse

    Sub init()
        mKundenID = 444
        mFirma = "DEF GmbH"
        mAdresse = "33333 Gütersloh"
    End Sub
End Module

Module Module3
    Sub kundenverwaltung()
        Call init() ' error, ambiguous
        Call Module1.init()
        Call Module2.init()

        Call zeigeKundenInfo(False)

        Module2.mKundenID = 333 ' error, private
        Module2.gLändercode = "DE"

        Debug.Print("Ländercode = " +
                    Module1.gLändercode)
        Debug.Print("Ländercode = " +
                    Module2.gLändercode)
        Debug.Print("Ländercode = " + gLändercode)
        ' error, ambiguous
    End Sub
End Module
```

Module - Ambiguities (Mehrdeutigkeiten)

```
' file1.vb
Module GlobalDeclarations1
    Public giVar%, gsVar!
    Sub globalProc() : End Sub
End Module

Module GlobalDeclarations2
    Public giVar%, gsVar!
    Sub globalProc() : End Sub
End Module

Module UseGlobalDefinitions1
    Private Sub useGlobalDefinitions()
        globalProc() ' error ambigous
        GlobalDeclarations1.globalProc() 'ok
        GlobalDeclarations2.globalProc()
        giVar = 1 ' error ambigous
        GlobalDeclarations1.giVar = 1 ' ok
        GlobalDeclarations2.giVar = 1
        GlobalDeclarations1.gsVar = 1
        GlobalDeclarations2.gsVar = 1
    End Sub
End Module
```

Modul - Ambiguities (Mehrdeutigkeiten)

```
' file2.vb
Namespace ScopeVisibility
    Public iVar% ' error
    Module GlobalDeclarations1 ' ok, now in a namespace
        Public globalVar%
        Sub globalProc() : End Sub
    End Module

    Module GlobalDeclarations2
        Public globalVar%
        Sub globalProc() : End Sub
    End Module
End Namespace

Module UseGlobalDefinitions2
    Private Sub useGlobalDefinitions()
        globalVar = 1 ' error, not visible
        ScopeVisibility.globalVar = 1 ' error ambigous
        ScopeVisibility.GlobalDeclarations1.globalVar = 1 ' ok, fully qualified
        ScopeVisibility.GlobalDeclarations2.globalVar = 1
    End Sub
End Module
```

Übung - Module

```
Namespace Abteilung ' file Abteilungen.vb
    Module Produktion
        Public abtName As String = "Produktion"
        Private Sub produziereArtikel(artikelID As Integer)
        End Sub
        Sub produziereArtikel(artikelID As Integer, stückzahl As Integer)
            For i = 1 To stückzahl
                produziereArtikel(artikelID)
            Next
        End Sub
    End Module
End Namespace

Module Technik
    Public abtName As String = "Technik"
    Const durchwahl As String = "-3499"
End Module

Module Personal
    Public Const durchwahl As String = "-6612"
End Module
```

Module

```
Module AppStart ' file Fakturierung.vb
    Sub main() ' Schreibweise ?
        neuerAuftrag()
    End Sub
End Module

Module Fakturierung
    Sub neuerAuftrag()
        WriteLine("neuer Auftrag an Abteilung " + abtName)            ' ?
        produziereArtikel(1001) ' produziere Artikel, Artikel-Nr=1001 ' ?
        Global.Microsoft.VisualBasic.Interaction.MsgBox(
            "bei technischen Problemen wählen sie: " + durchwahl)     ' ?
    End Sub
End Module
```

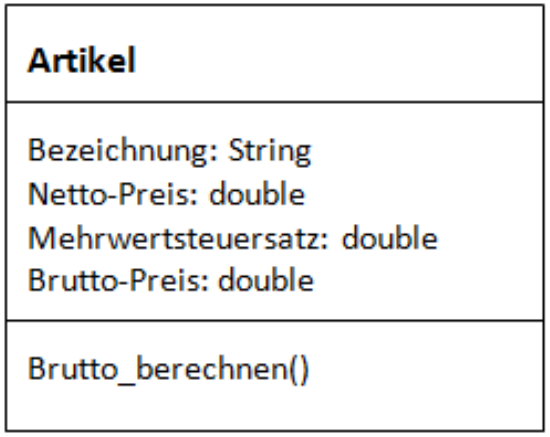
Functions

```
Module Classes
    Private Class Electronics
        Shared Function resistor#(ByVal ParamArray resistorValues#())
            Dim resTotal As Double
            For Each item As Double In resistorValues
                resTotal += 1 / item
            Next
            resistor = 1 / resTotal
        End Function
    End Class

    Sub Main()
        System.Console.WriteLine("Overall resistance={0:F2}",
                                 Electronics.resistor(1000, 5000, 3000, 4000))
                                 ' http://www.1728.org/resistrs.htm
    End Sub
End Module
```

Klassen

- ADT, abstrakter Datentyp
- kapselt Eigenschaften (Daten in Form von Variablen/Konstanten)
- kapselt Methoden (Funktionen auf den Daten)

- Klassenbezeichner, ClassID
- Attribute, Eigenschaften, Properties
- Methoden, Operationen



- Sichtbarkeit von Operationen und Attributen

  \+ ***public***, unbeschränkter Zugriff

  – ***private***, nur die Klasse selbst kann es sehen

Klassen - Konstruktor

- Initialisierung einer Klasseninstanz
- Standardkonstruktur u. weitere überladene Konstruktoren
- Vermeidung von Mehrdeutigkeiten im Konstruktor
  - Schlüsselwort Me: Verweis auf die aktuelle Instanz
  - Schlüsselwort MyClass: wie Me, überschriebene Methodenaufrufe werden ignoriert

```
Class A
    Private a, b, c

    Public Sub New(a, b, c)
        a = a ' ohne Effekt
        b = b
        c = c
    End Sub

    Public Sub New(a, b, c)
        Me.a = a ' Transfer
        Me.b = b
        Me.c = c
    End Sub

    Public Sub print()
        System.Console.WriteLine(a)
        System.Console.WriteLine(b)
        System.Console.WriteLine(c)
    End Sub

    Public Shared Sub Main()
        Dim instance As New A(1, 2, 3)
        instance.print()
    End Sub
End Class
```

Klassen - Zugriff innerhalb/außerhalb

- Zugriff auf Attribute einer Instanz innerhalb des Klassenkörpers mit Zugriffsspezifizierer
  - Private
  - Protected
  - Dim (wie Private)
- Main innerhalb der Klasse als Shared Main
  -
```
Class ClassName
    Public Shared Sub Main()
        Dim obj As New ClassName()
    End Sub
End Class
```

Klassen - Zugriff innerhalb/außerhalb

```
Class A
    Public a
    Private b
    Protected c
    Dim d ' wie Private

    Public Sub New(a, b, c, d)
        Me.a = a
        Me.b = b
        Me.c = c
        Me.d = d
    End Sub

    ' Zugriff innerhalb der Klasse (Main in der Klasse definiert)
    Public Shared Sub Main()
        Dim instance = New A(0, 0, 0, 0)
        instance.a = 1
        instance.b = 1 ' ok, access inside class A
        instance.c = 1 ' ok, access inside class A
        instance.d = 1 ' ok, access inside class A
    End Sub
End Class
```

Klassen - Zugriff innerhalb/außerhalb

```
Module MainModule
    ' Main im Modul definiert
    Sub Main()
        Dim instance = New A(0, 0, 0, 0)
        instance.a = 1
        instance.b = 1 ' error, not accessible
        instance.c = 1 ' error, not accessible
        instance.d = 1 ' error, not accessible
    End Sub
End Module
```

Klassen - Properties

- Für den vereinfachten Zugriff auf (Private) Attribute
- Definition als Methode, Aufruf durch Zuweisung
- Properties normalerweise als Public definiert (öffentliche Schnittstelle)
- Vgl. Auslesen der Informationen via Get-Methode
  - Schema
    ```
    Class A
      Private info
      Public Sub showInfo()
        System.Console.WriteLine(info)
      End Sub
    End Class
    ```
- als Property
  - Schema
    ```
    Class A
      Private _info
      Public Property info
        Get
          Return _info
        End Get
      End Property
    End Class
    ```
- Zugriff auf Attribut
  - Get-Methode: obj.showInfo()
  - Property: x = obj.info

Klassen - Properties

- Vgl. Setzen der Informationen via Set-Methode
  - Schema

```
Class A
   Private info
   Public Sub setInfo(newInfo)
      info = newInfo
   End Sub
End Class
```

- als Property
  - Schema

```
Class A
   Private _info
   Public Property info
      Set( value )
         _info = value
      End Set
   End Property
End Class
```

- Zugriff auf Attribut
  - Set-Methode: obj.setInfo(neuerWert)
  - Property: obj.info = x

Klassen - Properties

- Generelle Vorgehensweise
  1. evtl. bestehende Attribute umbenennen (ctrl + r + r)
     Attributsname → _Attributsname
  2. Property-Namen: wie ursprünglicher Attributsname
     Schema
     ```
     Public Property Attributsname As Datentyp
       Get
         Return _Attributsname
       End Get
       Set( value )
         _Attributsname = value
       End Set
     End Property
     ```
  3. Properties können auch automatisch generiert werden
     1. Attribute markieren, bzw. Cursor im Kontext eines Attributs
     2. rechte M. bzw. ctrl + . → Quick Actions.../encapsulate field(s)

Klassen - Properties

```
Class Customer
    Private _customerName As String
    Private _customerSince As Date

    Public Property customerName() As String
        Get
            Return _customerName
        End Get
        Set(ByVal Value As String)
            _customerName = Value
        End Set
    End Property

    Public Sub Capitalize()
        _customerName = UCase(_customerName)
    End Sub

    Public Sub Capitalize(decoration As Char)
        _customerName = Strings.StrDup(3, decoration) &
                        UCase(_customerName) &
                        Strings.StrDup(3, decoration)
    End Sub
```

Klassen - Properties

```
    Public Sub print()
        System.Console.WriteLine("Customer: " + _customerName)
        System.Console.WriteLine("Customer since: " + _customerSince)
    End Sub

    Public Sub New(ByVal customerName As String)
        _customerName = customerName
    End Sub

    Public Sub New(ByVal customerName As String, ByVal customerSince As Date)
        _customerName = customerName
        _customerSince = customerSince
    End Sub
End Class
```

Klassen - Properties

- nur Property Get
  - nur lesen
  - Modifizierer ReadOnly
- nur Property Set
  - nur schreiben
  - Modifizierer WriteOnly

```
Class A
    Public ReadOnly a
    Private _i
    Private _j

    Public ReadOnly Property i
        Get
            Return _i
        End Get
    End Property

    Public WriteOnly Property j
        Set(ByVal value)
            _j = value
        End Set
    End Property
```

Klassen - Properties

```
    Public Sub New()
        a = _i = _j = 1
    End Sub
End Class

Module MainModule
    Sub Main()
        Dim instance As New A()
        Dim val

        instance.a = 0 ' error, readonly
        val = instance.a

        val = instance.i
        instance.j = val
        val = instance.j ' error, writeonly
    End Sub
End Module
```

Klassen - Properties

- Property Get in Kombination mit Private Set
  - Objektebene nur lesbar
  - Instanzmethoden lesen u. schreiben
- Property Set in Kombination mit Private Get
  - Objektebene nur schreibbar
  - Instanzmethoden lesen u. schreiben

```
Class A
    Private _i
    Private _j

    Public Property i
        Get
            Return _i
        End Get

        Private Set(ByVal value)
            _i = value
        End Set
    End Property
```

Klassen - Properties

```
    Public Property j
        Private Get
            Return _i
        End Get

        Set(ByVal value)
            _j = value
            System.Console.WriteLine(_j)
        End Set
    End Property

    Private Sub s()
        Dim x
        x = i
        i = x
        x = j
        j = x
    End Sub
End Class
```

Klassen - Properties

```
Module MainModule
    Sub Main()
        Dim instance As New A()
        Dim val

        val = instance.i
        instance.i = val ' error, readonly

        instance.j = val
        val = instance.j ' error, writeonly
    End Sub
End Module
```

Klassen - Autoproperties

```
Class A
    ' autoimplemented properties
    ' generates private hidden field with Get/Set
    Property X As Integer                        ' hidden Field => _X
    Public Property Y As Integer                 ' hidden Field => _Y
    Public ReadOnly Property Z As Integer = 100 ' hidden Field => _Z

    Public Property PrimeValues As Integer() = {2, 3, 5, 7, 11, 13, 17}

    Private Sub S()
        Dim x = _X ' access to hidden field
        Dim y = _Y
        Dim z = _Z
        _PrimeValues(6) = 19
    End Sub
End Class

Module MainModule
    Sub Main()
        Dim instance As New A()
        instance.X = 1
        instance.Y = 1
        instance.Z = 1 ' error readonly
        instance.PrimeValues(6) = 23
    End Sub
End Module
```

Abstrakte Klassen

- Definitionen und Deklaration von Methoden u. Properties
- im Unterschied zu den Interfaces mit Implementierungen
- Instanziierung nicht möglich

```
Public MustInherit Class Kraftfahrzeug ' abstrakte Klasse
    Private strHersteller As String

    Public MustOverride Sub Starten() ' abstrakte Methode
    Public MustOverride Sub SystemCheck()
    Public MustOverride Property Motorleistung! ' abstrakte Property

    Public Function Anhalteweg(v As Double) As Double ' implementierte Funktion
        Anhalteweg = v * (v + 30) / 100
    End Function

    Public Property Hersteller As String ' implementierte Property
        Get
            Return strHersteller
        End Get
        Set
            strHersteller = Value
        End Set
    End Property
End Class
```

Abstrakte Klassen

```
Class Auto ' ctrl + . -> implement abstract class
    Inherits Kraftfahrzeug

    Public Overrides Property Motorleistung As Single
        Get
            Throw New NotImplementedException()
        End Get
        Set(value As Single)
            Throw New NotImplementedException()
        End Set
    End Property

    Public Overrides Sub Starten()
        Throw New NotImplementedException()
    End Sub

    Public Overrides Sub SystemCheck()
        Throw New NotImplementedException()
    End Sub
End Class

Module MainModule
    Sub main()
        Dim kfz As New Kraftfahrzeug ' error
        Dim a As New Auto()
    End Sub
End Module
```

Versiegelte Klassen

- Vererbungsfluß unterbrechen
- Schlüsselwort NotInheritable

```
NotInheritable Class Employee
End Class

Class Company
    Private employees(500) As Employee ' ok
End Class

Class BankEmployee
    Inherits Employee ' error
End Class
```

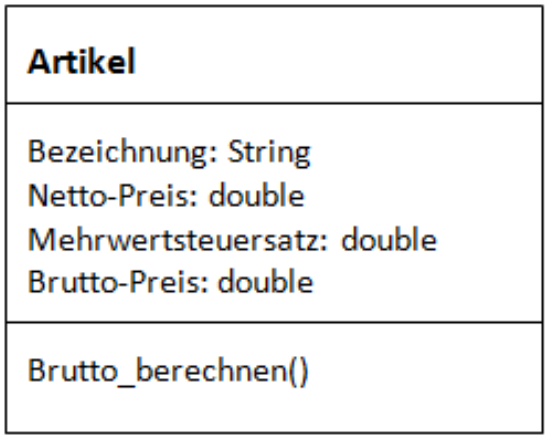
ADT - Übung

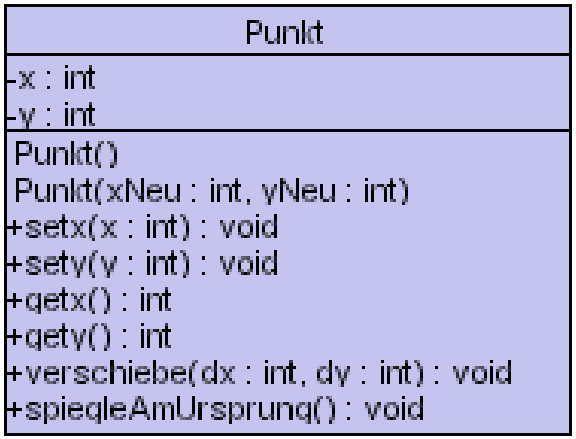
a) Definieren Sie die Klasse Grundstück mit den Komponenten Fläche, Preis und Ortsinfo(=konstant). Schreiben Sie zusätzlich einen Konstruktor und eine Ausgaberoutine. Leiten Sie daraus eine Feldvariable mit 5 Elementen ab und initialisieren Sie diese mit Werten ihrer Wahl. Geben Sie die Liste in einer Schleife aus.

b) Passen Sie Klasse Stack so an, dass Objekte der Klasse Grundstück aufgenommen werden können. Bei Über- oder Unterschreitung der Feldgrenzen soll eine 'ArrayIndexOutOfBoundsException' ausgelöst werden. Definieren Sie dazu einen weiteren Konstruktor, welcher die Feldlänge bestimmt. Im Fehlerfall soll die Meldung "Kein Speicher mehr verfügbar!" ausgegeben werden.

UML Klassendiagramm

- Klassenbezeichner, ClassID
- Attribute, Eigenschaften, Properties
- Methoden, Operationen





- Sichtbarkeit von Operationen und Attributen

  \+ ***Public***, unbeschränkter Zugriff

  \# ***Protected***, Zugriff nur von der Klasse sowie von Unterklassen(Klassen, die erben)

  – ***Private***, nur die Klasse selbst kann es sehen

Warum Vererbung?

- Anpassung/Erweiterung einer Klasse
  - erhöht Fehlerwahrscheinlichkeit, da bestehender (korrekter) Code verändert wird
  - vermindert Übersichtlichkeit
- Kopieren von bestehenden Deklarationen u. Definitionen mit anschließendem Umbenennen führt zu Redundanzen
- ⇨ Vererbungskonzept
  - neue Klassen aus bestehenden ableiten, damit bessere Wiederverwendbarkeit in anderem Problemkontext
  - Code der Basisklasse wird nicht verändert bei notwendiger Erweiterung der Funktionalität
  - Quellcode der Basisklasse muss nicht vorliegen (nur interpretierbare .class-Datei)
  - Vererbungshierarchie erlaubt mehrstufige Abstraktion, vom Allgemeinen zum Speziellen
  - stellt konsistente Änderungen für alle Unterklassen sicher
  - Objekte von Unterklassen sind wie Objekte von Oberklassen behandelbar, da sie die entsprechenden Eigenschaften geerbt haben

Beziehungen zw. Klassen

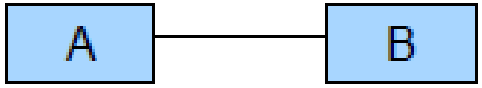
Hauptthema des objektorientierten Programmierens

Verwende bereits vorhandene Klassen, um neue Klasse zu definieren.
Baue Softwaresystem aus mehreren Klassen zusammen.

Dabei ergeben sich verschiedene Beziehungen zwischen den Klassen.

Beziehungen zwischen Klassen:

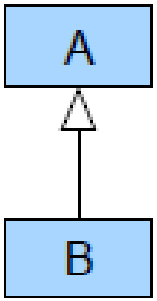
- **Assoziation**
  Beliebige Beziehung zwischen verschiedenen Klassen.



- **Aggregation**
  Spezielle Form der Assoziation:
  Ganzes-Teil-Beziehung oder Hat-Ein-Beziehung.

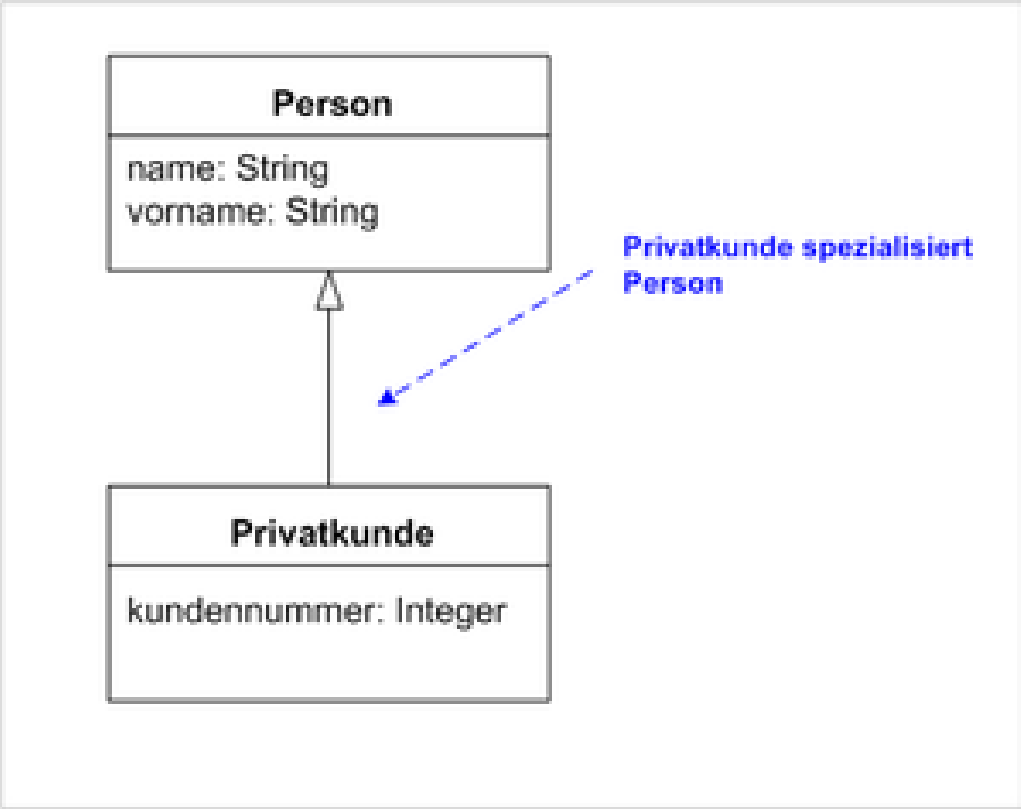
- **Komposition**
  Strenge Form der Aggregation,
  bei der die Teile vom Ganzen existenzabhängig sind.

- **Vererbung**
  - **Schnittstellenvererbung:**
    Ist-Ein-Beziehung
  - **Implementierungsvererbung:**
    Ist-Implementiert-Als-Beziehung



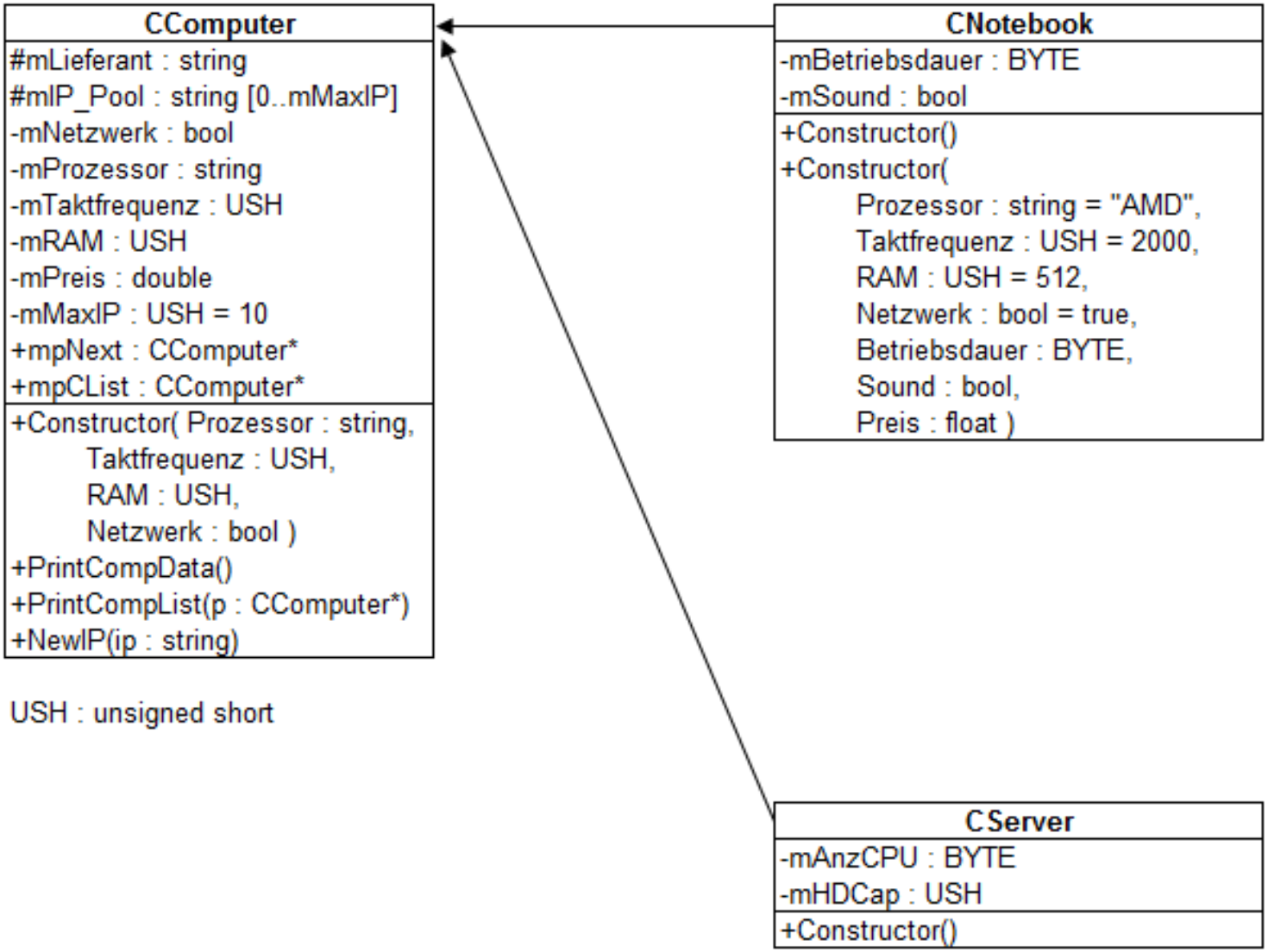
UML Klassendiagramm

- Generalisierung



UML Klassendiagramm

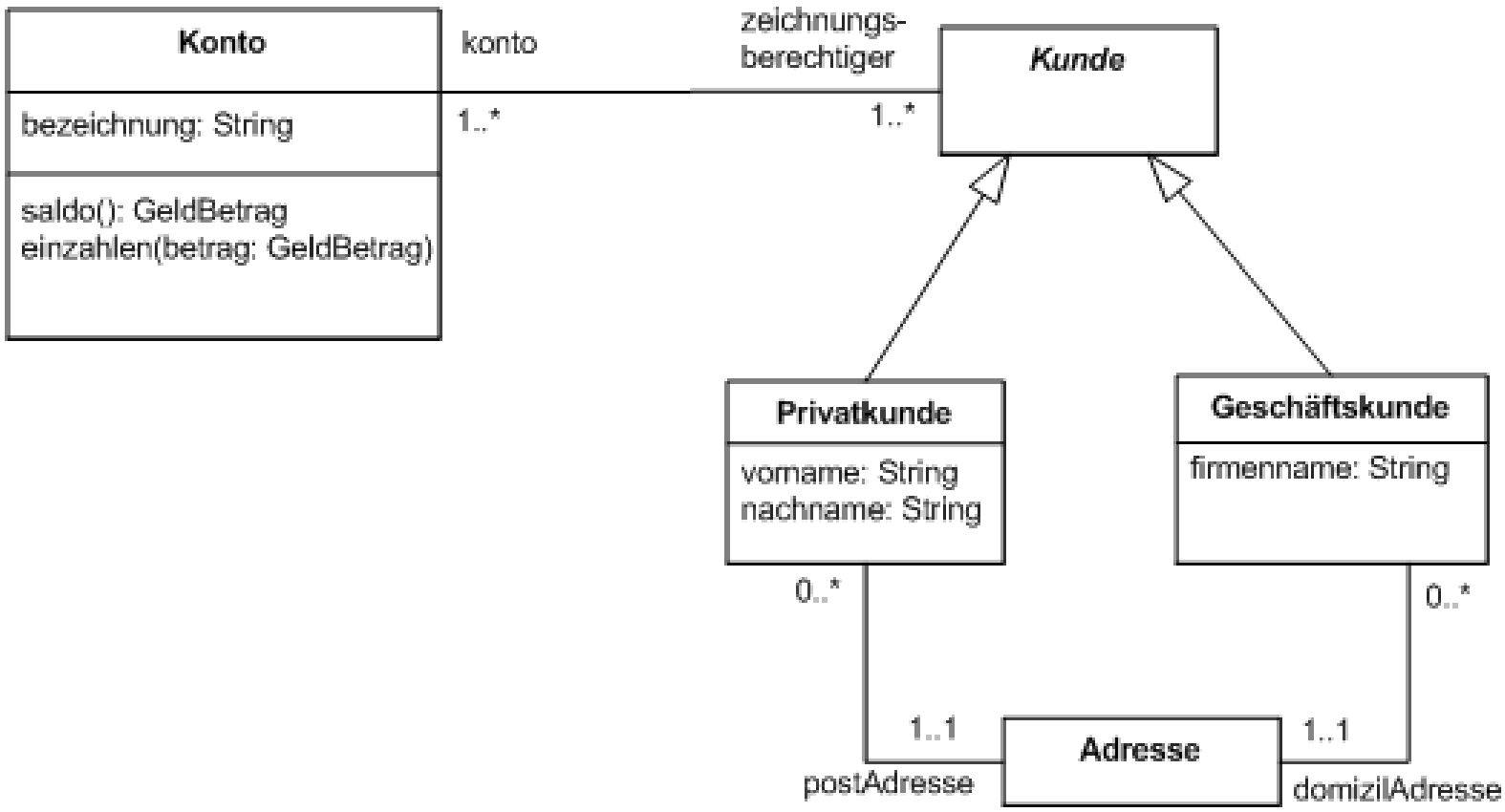
- Generalisierung



USH : unsigned short

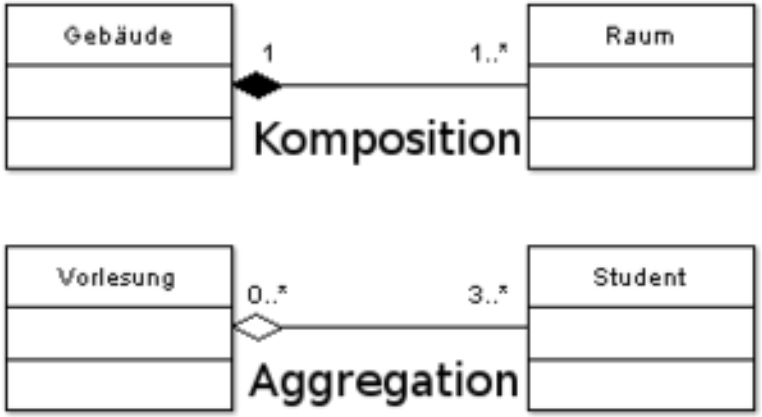
UML Klassendiagramm

- Assoziation, Beziehung zwischen Klassen, mit Angabe von Multiplizitäten



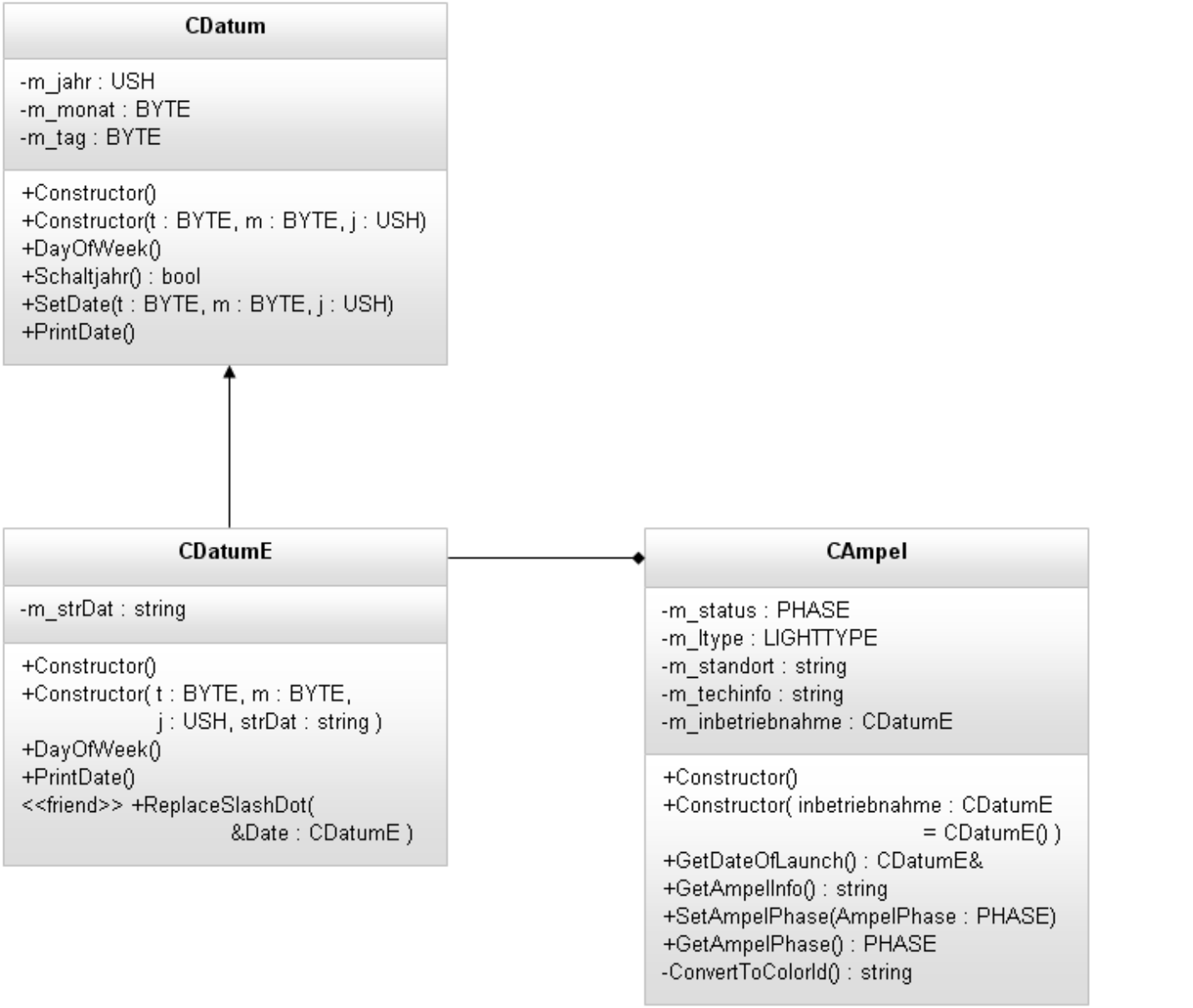
UML Klassendiagramm

- Komposition/Aggregation
  - Beziehung zwischen dem Ganzen und seinen Teilen
  - beschreibt "is part of"-Relation, Objekte haben einen stärkeren Zusammenhang als bei der Assoziation
  - Komposition ist Spezialfall der Aggregation
  - Rautensymbol am "Aggregat"



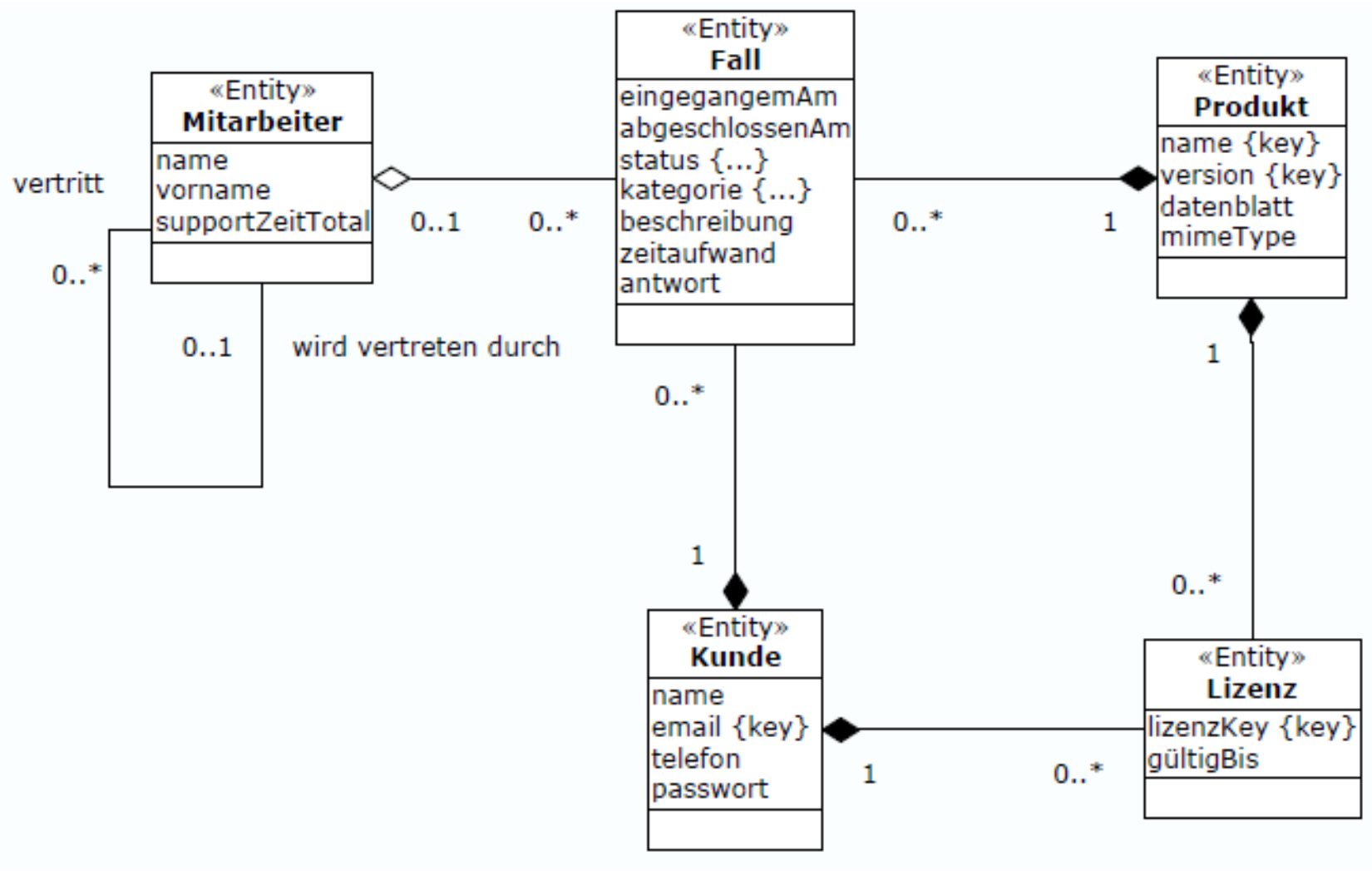
- Teile (Räume) können nicht ohne das Ganze (Gebäude) existieren (Existenzabhängigkeit)
- Teile (Studenten) können unabhängig vom Ganzen (Vorlesung) existieren

UML Klassendiagramm



PHASE : {ROT, GELB, GELBROT, GRUEN, BLINKEND}
LIGHTTYPE : {LED, BULB}

UML Klassendiagramm



- Aggregation: aggregiertes Objekt kann den Besitzer wechseln (Fallzuweisung an anderen Mitarbeiter)
- Komposition: Lizenz kann nicht auf anderen Kunden oder Produkt übertragen werden

Vererbung - Übung

a) Finden Sie eine Vererbungshierarchie bzw. geeignete Assoziationen für die folgenden Objekte:

Aspirin

Atom

Beruhigungsmittel

ChemElement

ChemVerbindung

Dosierung

Medikament

Wasser

Vererbung - Übung

b) Erstellen Sie einen Ableitungsbaum vom Allgemeinen zum Speziellen für die folgenden Objekte:

Parallelogramm

Quadrat

Raute (Rhombus)

Rechteck

Trapez

Vieleck

Viereck

Vererbung - Übung

c) Im Paket kreditwesen sind die untenstehenden Klassen definiert. Definieren Sie eine entsprechende Vererbungshierarchie mittels Einfach- und gegebenenfalls Mehrfachvererbung.

Häuslebauer

Arbeitnehmer

Kunden

Sparer

Bankkunde

Versicherungskunde

Kreditnehmer

VBA-Projekt - Aufbau

- Hierarchische Gliederung
- "TopLevel-Objekt": **VBAProjekt** (≙ Dateiname)
  - **Tabellenobjekte**
    (jd. Tabelle in der Arbeitsmappe entspricht einem Tabellenobjekt)
  - **Module**
    Container für Makros, benutzerdefinierte Prozeduren und Funktionen
    - Gültigkeiten
      - Procedure-level
      - Module-level
      - Global

Scope	Prefix	Example
Global	g	gstrUserName
Module-level	m	mblnCalcInProgress
Local to procedure	None	dblVelocity

      - alternativ: `g_`, `m_`
  - **Formulare**
    Eingabemasken für Datenerfassung, Darstellung von Daten, grafischen Objekten etc.
  - **Klassenmodule**
    objektorientierte Modulvariante

Modul - Aufbau

Microsoft Visual Basic for Applications - ExcelProjektMappe.xlsm

Datei Bearbeiten Ansicht Einfügen Format Debuggen Ausführen Extras Add-Ins Fenster ?

Frage hier eingeben

Z1, S1

Projekt - VBAProject

- VBAProject (ExcelProjektMappe.xlsm)
  - Microsoft Excel Objekte
    - DieseArbeitsmappe
    - Tabelle1 (Tabelle1)
    - Tabelle2 (Tabelle2)
    - Tabelle3 (Tabelle3)
  - Module
    - Modul1
    - Modul2

Eigenschaften - Modul2

Modul2 Modul

Alphabetisch | Nach Kategorien

(Name)	Modul2

ExcelProjektMappe.xlsm - Modul1 (Code)

(Allgemein) | (Deklarationen)

```
Public gLänderCode ' globale Variablen und Konstanten behalten ihren Wert auch nach _
                    Beendigung des Programms (außer bei Zurücksetzung des Projekts)
Private PLZ

Private Sub Postleitzahl()
    PLZ = 12345
End Sub

Sub PrintPostleizzahl() ' default public
    Call Postleitzahl    ' initialisieren
    MsgBox "Postleitzahl=" & PLZ
End Sub
```

Deklarationsteil

ExcelProjektMappe.xlsm - Modul2 (Code)

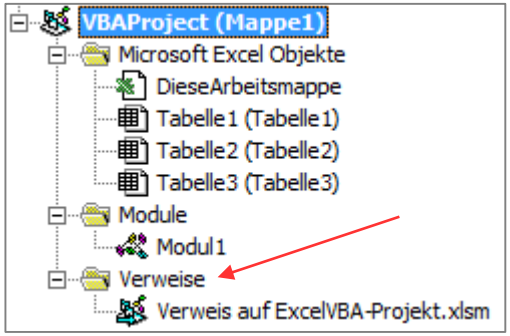
(Allgemein) | (Deklarationen)

```
Option Explicit

Sub LänderCode()
    gLänderCode = "DE"
End Sub

Sub NeueBestellung()
    Call Postleitzahl      ' nicht bekannt, da private
    Call PrintPostleizzahl ' ok
    MsgBox "Postleitzahl=" & PLZ ' Variable nicht definiert (nur bei Option Explicit)

    MsgBox "LänderCode=" & gLänderCode ' ok, aber nicht initalisiert
    Call LänderCode
    MsgBox "LänderCode=" & gLänderCode ' ok
End Sub
```

Makro Verteilung (Deployment) - Verweis

- Verweis
  - Bsp.: Quellmakromappe "ExcelVBA-Projekt.xlsm"

    11. Im Projektexplorer den Ordner "Verweise" prüfen



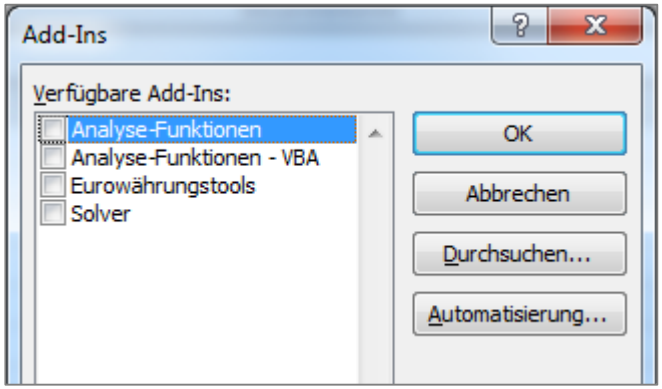
    12. Prüfen ob "TimerCode" kennwortgeschützt ist
    13. In der Prozedur "CallTimer" die Prozedur "Timer" aufrufen

```
Public Sub CallTimer()
    Call TimerCode.Timer  Call Timer würde Fehler liefern
End Sub
```

    14. In der Prozedur "CallTimer" die Prozedur "TimerSerial" aufrufen

```
Public Sub CallTimer()
    Call TimerSerial
End Sub
```

Makro Verteilung (Deployment) - Add-In



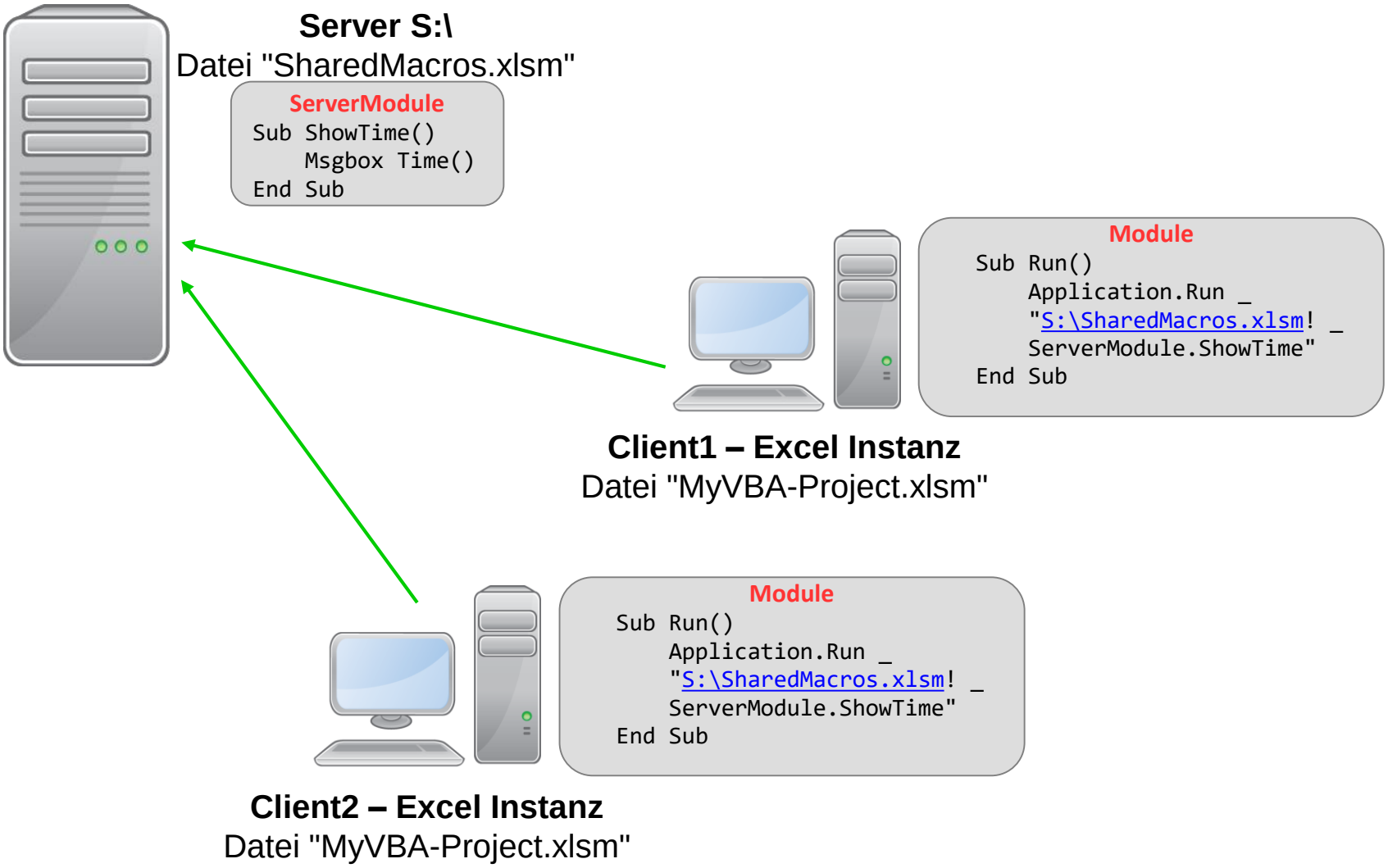
- Variante Add-In
  - Permanent eingebunden
  - Menü Entwicklertools/Add-Ins/Add-Ins bzw. Menü Datei/Optionen/Add-Ins
  - Anwenderseitig aufrufbar via Funktionsassistent
  - Programmierseitig nur über Fernbezug `Application.Run`
  - Bsp. Quellcodemappe als Add-In-Datei speichern

    1. Neue Mappe "Systeminfo"
    2. Zum VBA-Editor wechseln, ALT+E+M neues Modul erstellen
    3. Funktion "SystemInformationStart" als Hilfsfunktion, ALT+E+P+Function neue Funktion hinzufügen

```
Public Function SystemInformationStart()
    Call SystemInformation
    SystemInformationStart = 1
End Function
```

Makro Verteilung (Deployment) - Fernbezug

- Bsp.: Gemeinsam genutzte Quellcode-Datei auf Server



Message Box

ASCII	Konstante	Englisch	Deutsch
`Chr(9)`	`vbTab`	Tabulator	Tabulator
`Chr(10)`	`vbLf`	Line Feed	Zeilenvorschub
`Chr(13)`	`vbCr`	Carriage Return	Wagenrücklauf

Konstante	Wert	Beschreibung
`vbMsgBoxRight`	524288	Rechtsbündiger Text
`vbMsgBoxRtlReading`	1048576	Von rechts nach links lesen

Enum

- Aufzählungstyp ("enumeration")
- Datentyp zur Definition einer Menge an vordefinierten Konstanten
- Verbesserung der Lesbarkeit, Prüfung durch Compiler
- nicht typsicher

```
Public Enum DayType : SUNDAY : MONDAY : TUESDAY : WEDNESDAY : THURSDAY : FRIDAY
                    : SATURDAY : End Enum ' eine Programmzeile
Public Enum WorkDayType
    MONDAY
    TUESDAY
    WEDNESDAY
    THURSDAY
    FRIDAY
End Enum

Class Enumeration
    Public Shared Sub Main()
        Dim day As DayType ' defines a variable day of enumerated type DayType
        Dim workDay As WorkDayType
        day = DayType.SATURDAY ' set
        System.Console.WriteLine("today is: " & day) ' 6
        day = 77
    End Sub
End Class
```

Exceptions

```
On Error GoTo ErrHandler:
Worksheets("NewSheet").Activate
Exit Sub

ErrHandler:
If Err.Number = 9 Then
 sheet does not exist, so create it
Worksheets.Add.Name = "NewSheet"
 go back to the line of code that caused the
problem
Resume
End If
```

Exceptions

```
On Error Goto 0
On Error Resume Next
On Error Goto <label>:
```

```
On Error Resume Next
N = 1 / 0
If Err.Number <> 0 Then
        Debug.print Err.Number
        N = 1
End If

Exit sub/function

Label:
        Resume Next / exit

On Error GoTo Err1:
Debug.Print 1 / 0
 more code
Err1:
On Error GoTo Err2:
Debug.Print 1 / 0
 more code
Err2:
```

Pause

15

Mahlzeit!

60