Dauerauftrag:

* Die C# Schlüsselwörter lernen  
  <https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/>
* Die Operatoren lernen  
  <https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/operators/>
* Die Datentypen (Teil der Schlüsselwörter) lernen  
  <https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/types>
* Präprozessor-Direktiven lernen  
  <https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/preprocessor-directives/>

Meisinger Klemens  
WIFI OÖ GmbH

Einführung in die OOP mit C#

Termine:

Do 25.10

[Di 30.10](#ZweiteSitzung)

[Di 6.11](#DritteSitzung)

[Do 8.11](#VierteSitzung)

[Di 13.11](#FünfteSitzung)

[Do 15.11](#SechsteSitzung)

[Di 20.11](#SiebteSitzung)

[Do 22.11](#AchteSitzung)

Zeitplan:

17:30 – 18:45

19:00 – 20:15

20:30 – 21:45

Teilnehmerinnen und Teilnehmer

BAUER Angelika

BITZINGER Sarah

DULLER Verena

GEBETSROITHER Thomas

HUBER Bernhard

HUEMER Sebastian Christian

KRAGL Markus

LEITNER Niklas Matthias

PANHOLZER Andreas

PIRKLBAUER Hermann

RATZENBÖCK Martina

SCHLAGER Jürgen

SCHWARZ Dietmar

ZENKOVIC Damir

Inhalt:

* Grundbegriffe
* Analyse des Projekts
* Umsetzen der Algorithmen Bausteine mit C#
  + Sequenz
  + Verzweigungen
  + Schleifen
* Konstanten / Variablen / Fehlerbehandlung
* Eigene Ereignisse…

Im Training die Online Dokumentation von Microsoft:

* [docs.microsoft.com](https://docs.microsoft.com)

Was versteht man unter „Programmieren“?

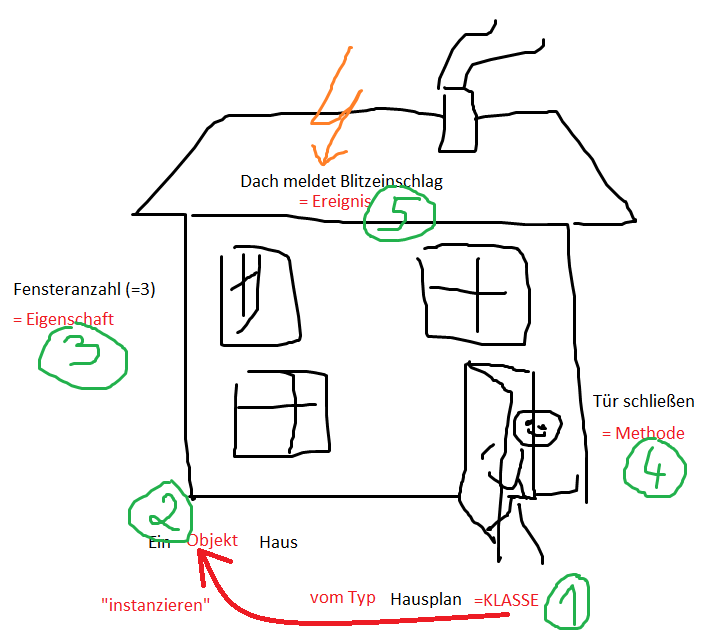
* Die Suche nach einer automatisierbaren Lösung für eine Aufgabenstellung
  + WICHTIG:   
      
    Programmieren ist „Nachdenken“!!!!!!
  + Auf keinen Fall: „Tippen“
* Beim Nachdenken, beim Programmieren, findet man den…  
    
  „Algorithmus“
* Das Schöne:
  + So ein Algorithmus hat nur drei Schritt:
    - Sequenz
    - Verzweigungen
    - Schleifen
  + Um 1970ig bewiesen, von  
    Nassi und Shneiderman  
      
    <https://de.wikipedia.org/wiki/Nassi-Shneiderman-Diagramm>

Für Neuprogrammiererinnen und Programmierer:

* Das erste Problemfeld:  
    
  Algorithmen finden (Nachdenken)

Das zweite Problemfeld:

* Es muss nicht auf „der grünen Wiese“ begonnen werden, wie früher
* Vieles ist schon fertig, z. B. wir müssen kein Betriebssystem programmieren, wir müssen keine Maussteuerung, … programmieren  
    
  Heute:
* „Objektorientierte Programmierung“

Grundbegriffe der „Objektorientierten Programmierung“ (OOP)  


Methoden können nur von Objekten benutzt werden.

Ausnahme: Vor dem Bau muss der „Hausplan genehmigt“ werden. „Genehmigen“ -> Verb, also Methode. Gehört aber nicht Objekt sondern der Klasse. So etwas nennt man „statisch“

1. Klassen, z. B. „Hausplan“
2. Zum Benutzen von Klassen  
   ist zu 99,999% der Fälle ein  
     
   Objekt  
     
   instanzieren.
3. Ein Objekt wird beschrieben mit  
     
   Eigenschaften
4. Zum Arbeiten mit Objekten, die  
     
   Methoden  
     
   aufrufen.
5. Für Mitteilung lösen die Objekte  
     
   Ereignisse  
     
   aus.

Viele Klassen sind heute bereits fertig. Zusammengehörende Klassen bilden ein…

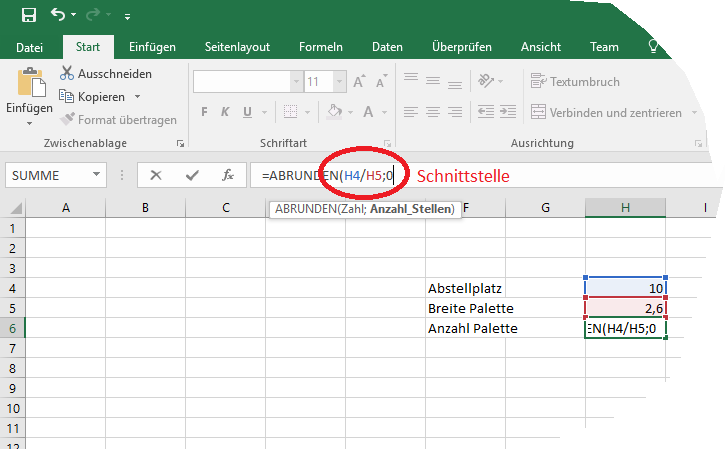
* Framework  
    
  z. B. Java Framework  
  Bei Microsoft:  
    
  -> .Net Framework

UNBEDINGT

* Die Konventionen vom .Net Framework einhalten

Zum Nachlesen:  
<https://docs.microsoft.com/de-de/dotnet/standard/design-guidelines/>

Nebenbei:

* Wie kommunizieren Methoden?
  + Über die Schnittstelle
  + Z. B. aus Excel bekannt  
      
    

„Die Schnittstelle“ von Funktionen, bei uns später „Methoden“

Zum .Net:

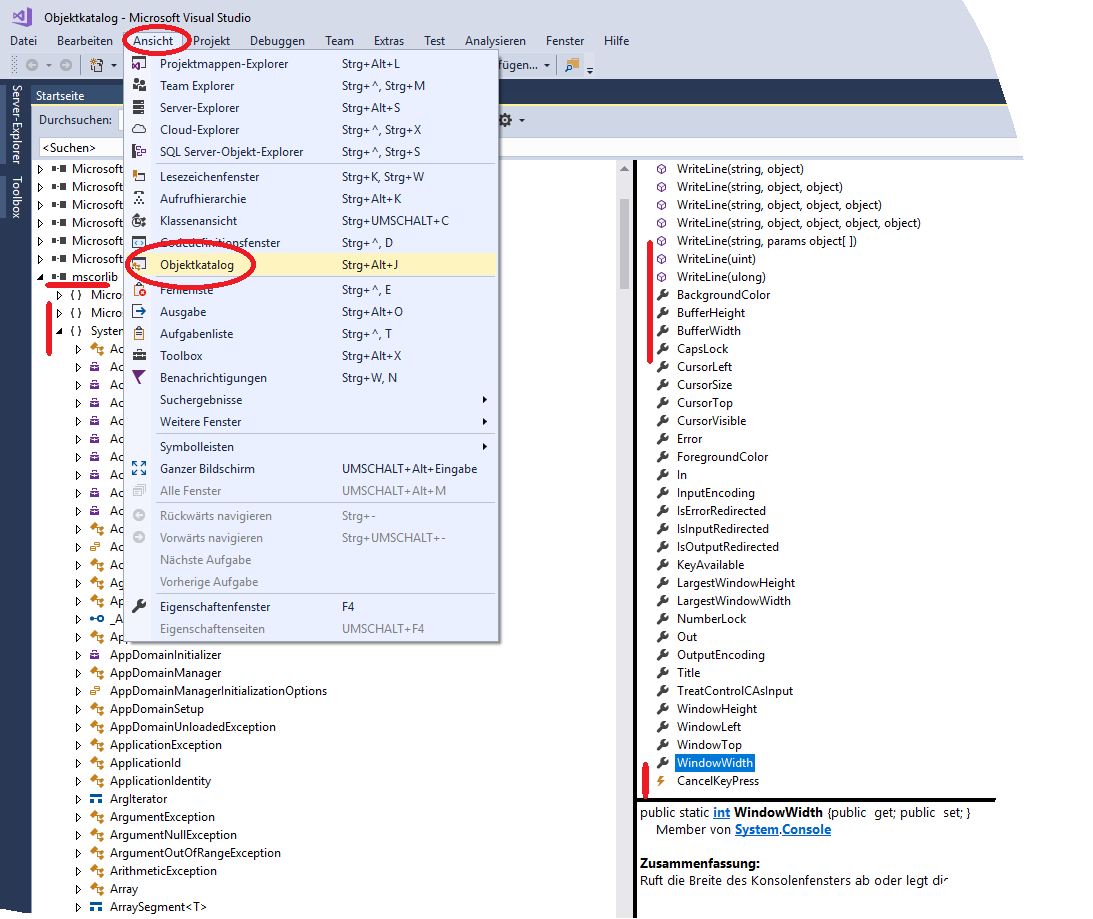
* Wie ist das gegliedert?

Auf dem Datenträger:

* Einzelne Dateien, die  
    
  Assembly
* In der Assembly eine Gliederung, die  
    
  Namespaces {}
* Im Namespace die …  
    
  Klassen  
  bzw. andere „Typen“
* In der Klasse:  
    
  - Eigenschaften  
  - Methoden  
  - Ereignisse

Wo kann man sich einen Überblick verschaffen?

* In der Entwicklungsumgebung



Ansicht / Objektkatalog (Strg + Alt + J)

1. Assembly: Firmenname.Problemlösung
   1. Namespace {}, langlebiger Bezeichner, versionsunabhängig
      1. Klasse, Substantiv, z. B. „Fenster“
         1. Eigenschaft, Schraubenschlüssel, Substantiv, z. B. „Zustand“
         2. Methode, Würfel, Verb, z. B. „Putzen()“  
            (Nur bei den Parametern in der Schnittstelle Kamel-Notation, d.h. das erste Zeichen klein, sonst überall „Pascal“, d.h. das erste Zeichen groß
         3. Ereignis, Blitz, Verbaler Ausdruck, z. B. „GlasBricht“

Die erste Assembly „mscorlib“ (Kernbibliothek). Der erste Namespace „System“. Die erste Klasse „Object“.

Wichtig:

* Derselbe Klassenname darf im gesamten Framework nur einmal vorkommen, d.h. muss eindeutig sein
* Um das zu gewährleisten:  
    
  Konventionen

Das war…

1. Problem  
     
   „Programmieren“, d.h. Nachdenken
2. Problem  
     
   „Fertige Lösungen“, das .Net Framework  
     
   Das Framework kann man nur verstehen.  
   Erfahrung sammeln. Suchen im Objektkatalog.

Wo ist C# (si-sharp)?

Das ist das dritte Problem:

* Beim Nachdenken findet man Algorithmen.
* Diese Algorithmen müssen  
    
  „getippt“  
    
  werden.
* Zum Tippen wird eine …  
    
  „Programmiersprache“   
    
  benötigt.

1. Generation der Programmierung:  
     
   Binärcode, 0 und 1 auf Lochkarten  
     
   z. B. 0010 1001 zum Addieren  
     
   war fad
2. Assembler – Code  
     
   Kürzel für die Bit-Kombinationen, z. B. „ADD“ für „0010 1001“  
     
   Damit der Prozessor den Assembler Code hat ausführen können, mussten die Kürzel wieder in 0 und 1 übersetzt werden. Dazu waren eigene Tools notwendig, die „Compiler“  
     
   war fad
3. Höhere Programmiersprachen  
     
   Hier wurde „angelehnt an natürliche Sprachen“ getippt  
     
   z. B. Cobol. Hier hat eine Anweisung mit einem Punkt „.“ geendet.  
   Weitere Sprachen hier…
   1. C -> ohne Objektorientierung
   2. C/C++ -> mit Objektorientierung  
      (Heute extrem wichtig in der Systemprogrammierung)
   3. PASCAL
   4. BASIC
   5. FORTRAN
   6. …
   7. Mit .Net wurde eine neue Sprache von  
      Microsoft eingeführt:  
        
      C#
4. Generation  
   „Grafische Programmierung“, z. B. in der Maschinenautomatisierung  
   oder „Adobe Flash“ und … grrrrrr

C# ist cool…

* Warum?
  + Weil C# nix kann
  + C# kann nur…
    - Namespaces
    - Klassen
    - Eigenschaften
    - Methoden
    - Ereignisse
    - UND die Algorithmenbaustein
      * Sequenz
      * Verzweigungen
        + „Binärverzweigung“
        + „Fallentscheidung“
      * Schleifen
        + „Zählschleife“, wenn die Anzahl der Wiederholung bekannt ist
        + „Abweiseschleife“, wenn die Anzahl der Wiederholungen vollkommen unbekannt ist, unter Umständen sogar nie
        + „Durchlaufeschleife“, wenn die Anzahl der Wiederholungen unbekannt ist, aber mind. 1x
* Dazu gibt’s die „Schlüsselwörter“
* Diese haben auf einer Seite Platz  
  <https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/>
* Diese sind ZUM AUSWENDIGLERNEN!!!!!!!!

Und von wo kommen die viel gepriesen Möglichkeiten?

* Nicht von C#, vom .Net Framework

Zwei Möglichkeiten, ein Training durchzuführen:

* Erste Möglichkeit (alle Bücher sind so aufgebaut)
  + Erster Abend: Neues Projekt, Sequenz
  + Zweiter Abend: Neues Projekt, Binärentscheidung
  + …  
      
    Hier lernt man keine objektorientiere Programmierung
* Zweite Möglichkeit (hier)
  + EIN Projekt, in dem alle Punkte aufgehen
  + Objektorientiert

Auftrag:

* Eine Anwendung ist zu erstellen, wo einer Person, die noch nie programmiert hat, gezeigt wird, wie mit C# objektorientierte Lösungen aufgebaut und die Algorithmen-Bausteine umgesetzt werden.

Fertigstellung: 22. November 2018

Womit beginnt ein Projekt?

* Mit der Analyse
* Heute objektorientiert
* Beschrieben mit der „Unified Modeling Language“ (UML)  
  <https://www.ibm.com/developerworks/rational/library/769.html?mhq=uml%20basics&mhsrc=ibmsearch_a>
* Diese UML ist Standard in der Software Architektur seit ca. 2000
* Ergebnis der Analyse:  
    
  K L A S S E N D I A G R A M M

Der Weg dorthin ist ein langer…

* Hier werden erfahrene Leute benötigt

Die wichtigsten Punkte der UML für unser Projekt:

1. The Actors Diagram  
     
     
   „Neuprogrammierer“  
     
   Bei echten Projekten, kann das „explodieren“, d.h. viele Personen, Personengruppen oder andere Systeme
2. The Use Cases Diagram (je Actor)

Die Sequenz



Die Zählschleife

Die Binärentscheidung

Die Abweiseschleife

Die Fallentscheidung

Die Durchlaufeschleife

Hinweis:

* Zum Abbilden der UML sind eigentlich spezialisierte Programme notwendig, z. B. Microsoft VISIO, oder das kostenpflichtige Visual Studio

1. The Szenario Diagram (je Use Case)

|  |  |
| --- | --- |
| Anwendungsfall | Beschreibung |
| Die Sequenz | Den Text „Hallo Welt!“ auf den Bildschirm zaubern |
| Die Binärentscheidung | Ein zufällige Ganzzahl zwischen 1 und 100 würfeln und entscheiden, ob die Zahl unter 50 oder größer gleich 50 ist. |
| Die Fallentscheidung | Für eine zufällige Stunden einen Begrüßungstext ermitteln. |
| Die Zählschleife | Die Begrüßung aus der Fallentscheidung für jede mögliche Stunde testen (von 0 bis 23) |
| Die Abweiseschleife | Den Inhalt einer unformatierten Textdatei lesen und am Bildschirm ausgeben. |
| Die Durchlaufeschleife | Zufällig 6 unterschiedliche Ganzzahlen zwischen 1 und 45 ermitteln (Lotto-Problem) |

… die UML kennt jetzt noch zahlreiche weitere Darstellungen. Für uns reicht das hier

Zum Ergebnis der Analyse:

* Klassendiagramm

Wie zeichnet die UML eine Klasse?

|  |
| --- |
| Klassenname |
| Eigenschaften |
| Methoden |
| Ereignisse |

Wie findet man die Klassen?

Weil wir eine „Anwendung“ erstellen, sicher die…

„Anwendung“ – Klasse

Zur viel diskutierten Sprache:

* Die WIFI Firmensprache ist „Deutsch“  
    
  Falls die Firmensprache „Deutsch“, dann „Deutsch“ für die Klassen, Eigenschaften, Methoden und Ereignisse
* Bei Fronius „English“  
  Hier „Englisch“. Hier muss aber FÜR ALLES ein englischer Ausdruck vorhanden sein  
    
  NIE, NIE die Sprachen „mischen“

Neben der „Anwendung“, eine „Algorithmus“ – Klasse

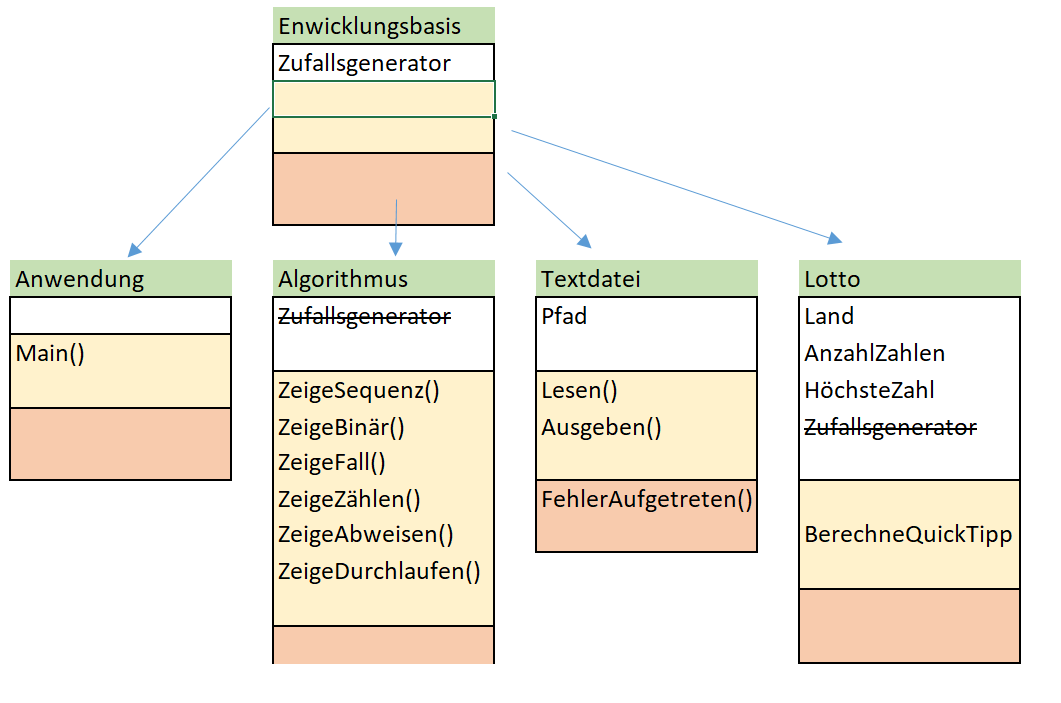
Je nach Komplexität eigene, zusätzliche Klassen.

Eine Klasse soll immer für „eine Aufgabe“ zuständig sein (in sich geschlossen)

Deshalb zusätzlich…

„Textdatei“ und „Lotto“ Klasse

Rohentwurf



Die Abbildung zeigt, welche Klassen mind. vorhanden sein werden.

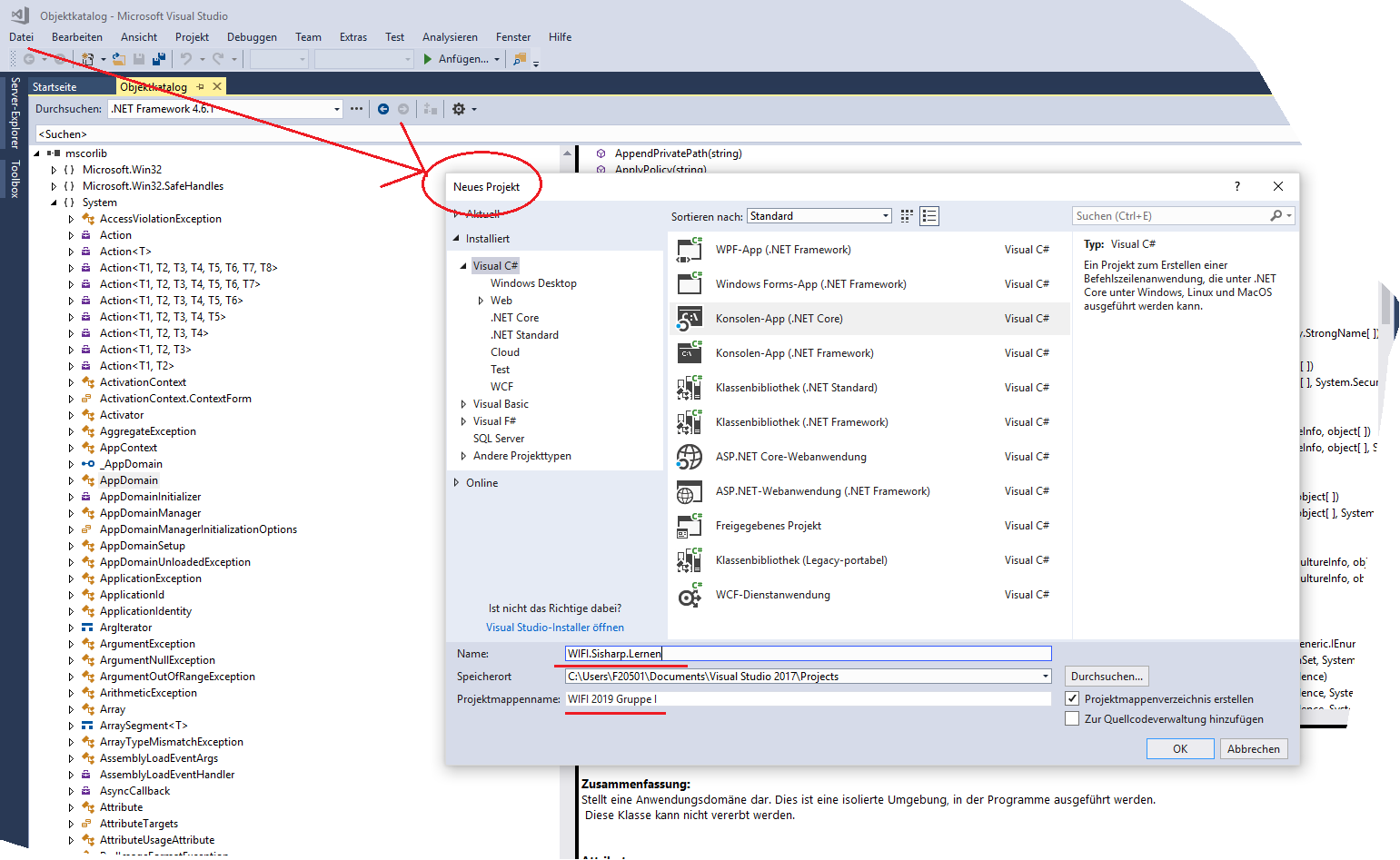
Tipp:

Wichtig:

* Zur Frage, ob richtig oder nicht richtig
* Grundsätzlich:
  + So viele Programmiererinnen und   
    Programmierer, so viele Lösungen
  + NIE in der objektorientierten Programmieren „eine alte, andere Klasse öffnen, Teil vom Code kopieren und in einer neuen, anderen Klasse einfügen“ -> Fehler  
      
    Die Teile, die mehrfach benötigt werden, gehören in eine „Basisklasse“  
      
    (Objektorientierte Programmierung MIT VERERBUNG)
* .Net unterstützt nur die Einfachvererbung, d.h. eine Klasse kann nur eine Basisklasse besitzen (im Gegensatz zu Objectiv-C, die Apple Programmiersprache zum Entwickeln für MacOS)  
  Hinweis zum Apple:   
  Dort läuft mittlerweile ebenfalls Microsoft Visual Studio

Zum Umsetzen:

* Ein Projekt einrichten
* Ergebnis ist eine Assembly
* Dazu ein Name  
    
  WIFI.Sisharp.Lernen  
    
  Schwerpunkt liegt auf den Schlüsselwörtern von C#, nicht am .Net  
  Deshalb eine Anwendung ohne grafischer Oberfläche (d.h. Konsole)



Die Abbildung zeigt, wie über Datei / Neu / Neues Projekt eine C# Konsolenanwendung mit der Bezeichnung WIFI.Sisharp.Lernen in einem Ordner (Projektmappe) „WIFI 2019 Gruppe I“ erstellt wird.

Unterschied zwischen

* .Net Core
* .Net Framework

Mit 1999 hat Microsoft „.Net Framework“ auf den Markt gebracht. Unterschied zu Java. Das .Net Framework kann nur unter Windows Betriebssystemen benutzt werden.

* Vieles, vor allem im Internet Bereich läuft unter Linux
* Microsoft versucht, das .Net Framework auch unter Linux zum Laufen zu bringen.  
    
  Die Betriebssystem unabhängigen Teile sind „.Net Core“

30.10.2018

Bisher:

1. Programmieren = Nachdenken  
   Das Ergebnis: Algorithmus  
   Nur drei Bausteine
   1. Sequenz
   2. Entscheidungen
      1. Entweder-oder (Binär)
      2. Fallentscheidung
   3. Schleifen
      1. Zählschleife
      2. Abweiseschleife  
         (kopfgesteuerte Bedingungsschleife)
      3. Durchlaufeschleife  
         (fußgesteuerte Bedingungsschleife)
2. Tippen  
     
   Programmiersprache notwendig  
   -> C# (si-sharp)
3. Fertige Lösungen  
   (als Klasse getippt)  
   Zusammengehörende Klassen bilden ein Framework.  
   Bei uns: .Net Framework

Aufbau:

Am Datenträger: Assembly

In einer Assembly

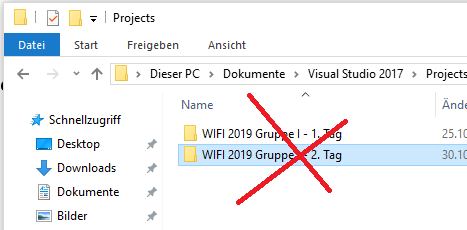
* Namespaces
  + Klassen
    - Eigenschaften
    - Methoden
    - Ereignisse

Hier:

* Ein Projekt

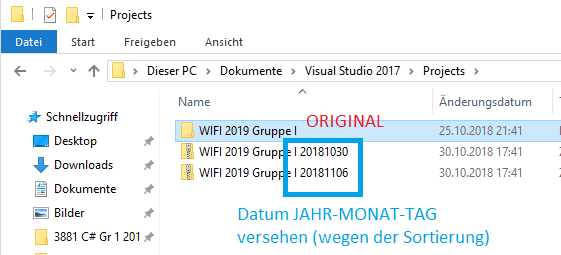
WICHTIG:

Auf keinen Fall …



… den Projektordner ändern. Nicht 1. Tag, 2. Tag …

Die Idee mit dem Sichern ist ja OK. Aber richtig.



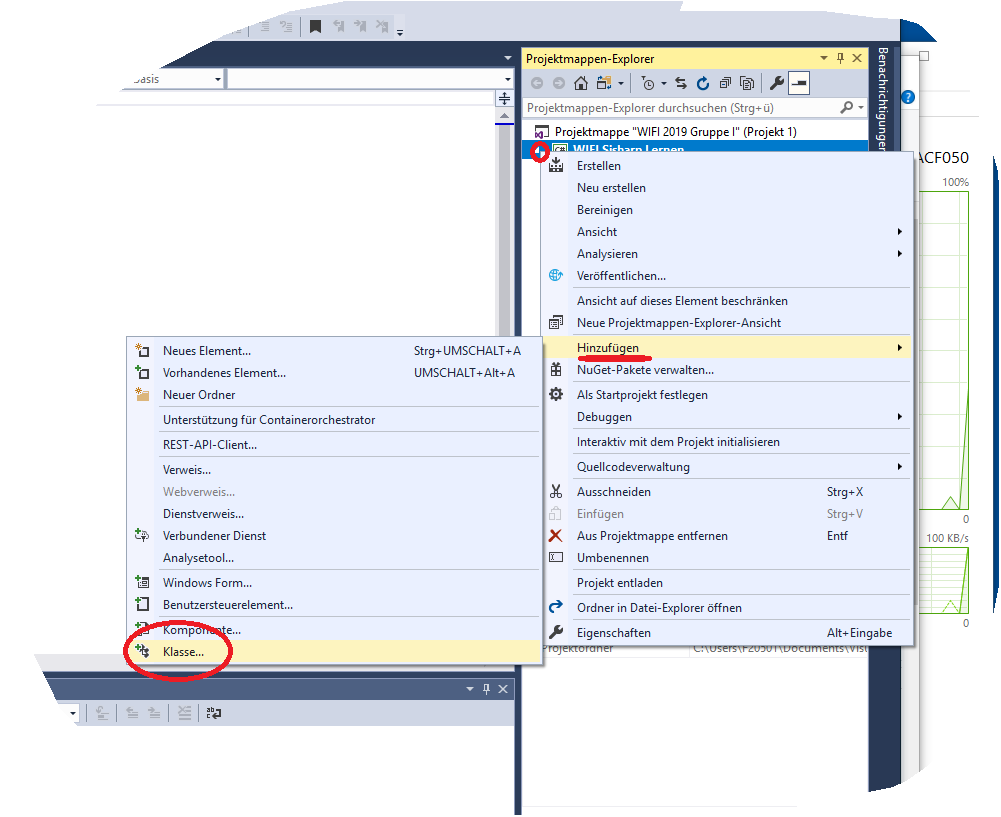
Am besten…

1. Das Original zippen
2. Das Zip benennen. Am besten  
   mit Datum in der Form JJJJMMTT

Der erste Schritt bei einem neuen Projekt:

* Die Analyse
* Heute objektorientiert
* Beschrieben mit der UML
* Ergebnis der Analyse:  
    
  Das Klassendiagramm

Umsetzen des Klassendiagramms



Projekt / Kontextmenü / Hinzufügen / Klasse…

* Entsprechend der Konventionen benennen!

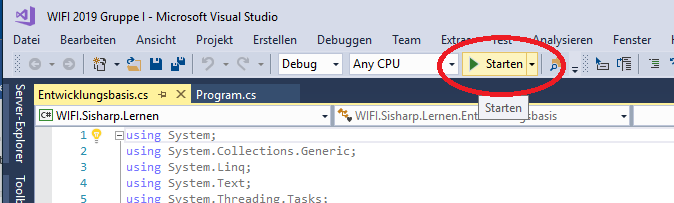
Fragen:

* Hilfe, bei mir fehlt der Projektmappen-Explorer.   
  Wie finde ich diesen wieder?
  + Menü Ansicht
  + Projektmappen-Explorer

Tipp:

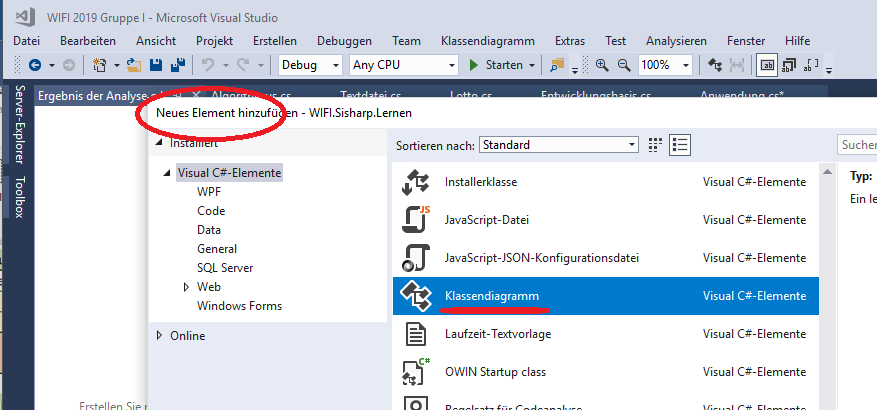
* Visual Studio kennt den Befehl Fenster / Fensterlayout zurücksetzen
* Wie kann ich auf eine Sicherung zurückgreifen?

1. Studio schließen
2. Den Original Ordner löschen (damit fehlerhafte Dateien  
   auch wirklich weg sind)
3. Aus dem Sicherungs-Zip den Ordner kopieren  
   und als Original einfügen
4. Projekt im Studio wieder öffnen

* Wann muss ich im Studio speichern?
  + Im Grunde nie
  + Studio speichert automatisch, wenn  
      
    das Projekt gestartet wird
  + Nur, wenn das Projekt längere Zeit nicht gestartet wird,  
    die Symbole zum Speichern benutzen.

Tipp:

Visual Studio kann das Ergebnis der Analyse …



… als „Klassendiagramm“ darstellen. Die Abbildung zeigt, wie über   
„Hinzufügen / Neues Element…“ ein Klassendiagramm erstellt wird.

* Nur verfügbar, wenn die Komponente „Klassen-Designer“   
  installiert wurde

Wenn in C# getippt wird, wird in Wirklichkeit…

* .Net Entwicklung

… betrieben.

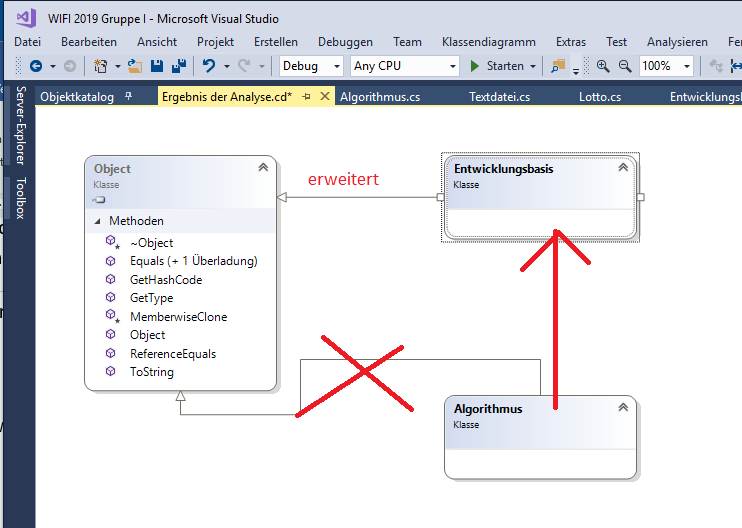
* Wo sieht man die .Net und eigenen Klassen?

Im Objektkatalog (Ansicht oder Strg + Alt + J)

* Erste Microsoft Assembly: mscorlib
* Erster Namespace: System
* Erste Klasse: Object

WICHTIG:

* ALLES Eigene ist eine Erweiterung von  
  System.Object



Beim Hinzufügen der eigenen Klassen wird sofort erkennbar, dass alles eine Erweiterung von System.Object ist.

Tipp:

* Eigene Projekte nicht direkt auf System.Object aufsetzen
* Warum?
  + Von System.Object existiert kein Programmtext (für uns)
  + Es kommt immer wieder vor, dass später für mehrere Klassen „identische“ Mitglieder (Eigenschaften, Methoden, Ereignisse) benötigt werden
  + Mit einer eigenen Entwicklungsbasis bleibt man flexibel

Wie wird in C# eine andere Basisklasse als System.Object festgelegt?

* Tippen
* Schlüsselwörter kennen

Alle C# Schlüsselwörter sind klein geschrieben!

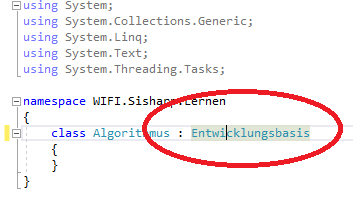
Die Groß-/Kleinschreibung muss beachtet werden!  
(C# ist case-sensitiv, im Gegensatz zu BASIC)

Für die Vererbung:

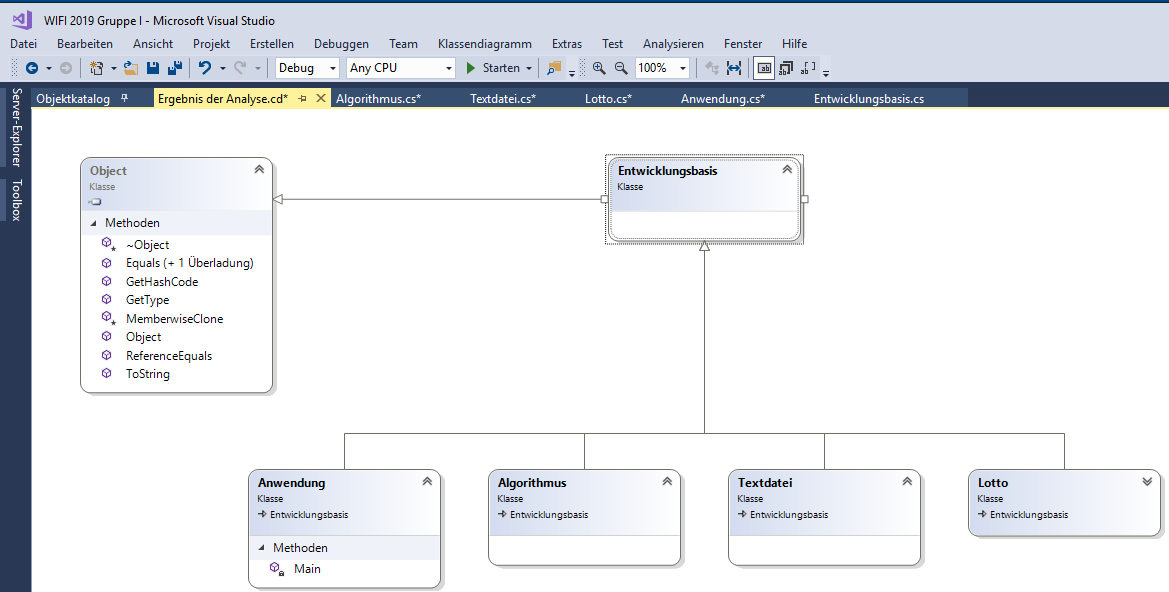
* Nur Einfachvererbung in .Net
* Syntax in C#  
    
  class MeineKlasse : BasisKlasse  
  {  
  }

Für einen Gültigkeitsbereich nutzt C#, nicht wie BASIC mit „END Methode“

die geschweiften { } Klammern.

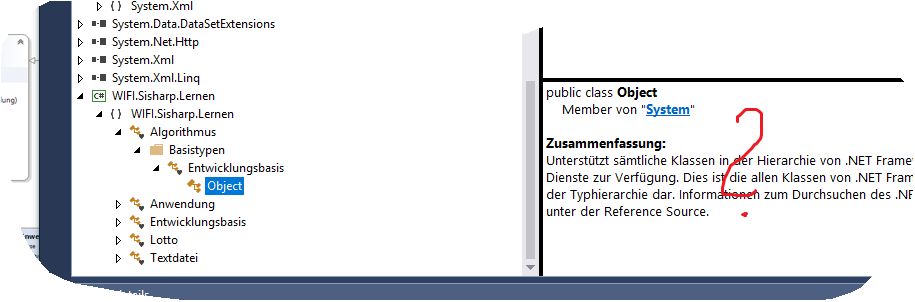


Die Abbildung zeigt, wie in C# eine andere Basisklasse als System.Object genutzt wird.



Die Abbildung zeigt das Klassendiagramm mit allen Klassen   
entsprechend der Analyse.

Nach dem Festlegen der Klassen …



… das Wichtigste:

* Es darf NICHTS, wirklich NICHTS ohne Kommentar geben

DAS KOMMENTAR unbedingt AM ANFANG der Tipperei (und nicht am Ende)

Mit .Net hat Microsoft eine neue Technologie zum Kommentieren eingeführt:

* Xml-Dokumentationskommentar

Alter C-Kommentar:

/\*

Hier die Doku

\*/

Mit C/C++ wurde …

// Hier die

// Doku…

Im Gegensatz zum C – Kommentar wird der C/C++ Kommentar   
automatisch am Zeilenende beendet.

Mit .Net, C#

/// <xmlTag>

/// Hier die Doku

/// </xmlTag>

wurde Xml-Dokumentationskommentar eingeführt.

C# unterstützt alle Versionen zum Kommentieren.

Vorteil:

* Dieser wird automatisch im Objektkatalog  
  und beim Tippen angezeigt

Handbuch:

<https://docs.microsoft.com/de-de/dotnet/csharp/programming-guide/xmldoc/xml-documentation-comments>

Was steht im Kommentar?

* Im Kommentar steht, was man beabsichtigt zu tun  
  Im Grunde die Aufgabenstellung
* NICHT: Wie man etwas macht

Bisher wissen wir:

* Analyse
* Klassendiagramm
* Im Studio ein Projekt anlegen
* Klassen erstellen
* Vererbung
* KOMMENTARE!!!!!!!!

Außerdem gesagt:

* Eine Klasse ist vergleichbar mit einem Bauplan
* In diesem Bauplan kann nicht gewohnt werden
* Zum Wohnen das Objekt „Haus“ erstellen

Zur „Entwicklungsbasis“

* Ist nur eine Grundlage, aber soll nicht   
  als Objekt benutzt werden können

Lösung:

* Modifizierer in C#

Handbuch

<https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/modifiers>

Wichtige Modifizierer sind…

* Für den Gültigkeitsbereich (Zugriffsmodifizierer)

Grundprinzip der objektorientierten Programmierung:

* Alle ist (standardmäßig) PRIVAT
* Freiwillig gibt man nichts her
* Ursache: Sicherheit

Deshalb:

* „internal“ Standard bei Klassen
* D.h. nur innerhalb der Assembly,   
  aber nicht außerhalb nutzbar

Wir benötigten aber die Entwicklungsbasis im C# Teil 2…

* Deshalb „public“

Manchmal soll verhindert werden, dass eine Klasse  
als Basisklasse benutzt werden kann.

(Widerspricht im Grunde der objektorientierten Programmierung)

Beispiel:

* C# hat (wie alle .Net Sprachen) keine eigenen  
  Datentypen (zum Reservieren von Speicher)
* Alle Typen kommen von .Net  
  Beispiel: Arbeiten mit Text über System.String
* Diese Klasse ist „sealed“ modifiziert, d.h.  
  System.String kann nur als Objekt benutzt werden,  
  aber keine Basisklasse sein

Nächstes Thema:

* Wie wird mit C# eine Methode implementiert?

1. Mit Rückgabe: Hier wird ein Datentyp benötigt
2. Ohne Rückgabe: Schlüsselwort „void“

6.11.2018

Fragen:

* Warum funktioniert bei mir der Code von einem Kollegen nicht, wenn diesen in mein Projekt kopiere – obwohl‘s eigentlich derselbe ist?  
    
  Ursache kann nur die Groß-/Kleinschreibung sein  
  C# ist case sensitiv!
* Warum werden die Mitglieder Zeige… in der privaten „internal“ Klasse „public“ modifziert?

Weil wir innerhalb der Assembly mit ZeigeSequenz(), ZeigeBinär(), … arbeiten wollen. Wären diese Methoden in der Algorithmus Klasse „private“, könnten sie nur innerhalb von Algorithmus benutzt werden, aber nicht in der Anwendung.

* Wofür steht „void“?  
    
  Zwei Möglichkeiten von Methoden
  + Methoden, die etwas arbeiten, aber nichts zurückgeben
    - In BASIC wäre das „Sub“
    - In allen anderen Sprachen Methoden   
      mit dem Schlüsselwort „void“
  + Methoden die etwas berechnen und das Ergebnis zurückgeben, das sind die Funktionsmethoden
    - In BASIC wäre das „Function“
    - „Function“ Schlüsselwort gibt’s in C ähnlichen Sprachen nicht. Hier muss „void“ durch den Rückgabedatentyp ersetzt werden
* Was passiert, wenn ich bei einer Methode keinen Zugriffsmodifizierer (private, public oder protected) angebe?
  + Standard
  + Standard in der objektorientierten Programmierung: private
* Was sind „Datentypen“?
  + Haben wir bisher noch nicht besprochen
  + Grundsätzlich werden diese zum Reservieren von Arbeitsspeicher benutzt

Neu:

* Alle unsere Methoden Zeige… müssen etwas auf dem Bildschirm ausgeben. Das Ausgeben wird auch in der Anwendung und anderen Klassen benötigt, …
* Deshalb:  
    
  Eine Methode zum Ausgeben in der Entwicklungsbasis
  + Damit die Methode nicht im Objekt sichtbar ist, nicht „public“
  + „private“ würde keinen Sinn ergeben, weil dann die Methode nur innerhalb der Entwicklungsbasis genutzt werden kann. Wir aber die Methode in die Entwicklungsbasis verschieben, damit diese in allen abgeleiteten Klassen sichtbar ist
  + Mittelding zwischen „public“ und „private“  
      
    „protected“  
      
    Protected-Mitglieder sind in abgeleiteten Klassen   
    nutzbar, nicht aber im Objekt

Außerdem benötigten reservierten Arbeitsspeicher:

* Zum Reservieren von Arbeitsspeicher
  + Die Datentypen
  + Diese gehören nicht C#
  + Diese gehören .Net
  + C# hat nur „Abkürzungen“ (aliases) zu den .Net Typen  
    (Teil der Schlüsselwörter)  
    <https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/types>

Grundsätzlich wird bei Daten zwischen Text, Zahlen, … unterschieden.

z. B. für einen Text  
System.String

für JA (true) oder NEIN (false)  
System.Boolean

für ein 32 Bit Ganzzahl

System.Int32

…

Grundsätzlich werden die Datentypen in …

* Zwei Familien

… eingeteilt.

1. Die Wertypen, wo die Daten „direkt“ im Namen gespeichert sind  
     
   Das sind die Strukturen („struct“)
2. Die Verweistypen. Hier befindet sich nur eine Speicheradresse „im Namen“, die auf die Daten „zeigt“ (verweist)  
     
   Das sind die Klassen („class“)

Grundsätzlich:

* Mit einer Klasse macht man nie einen Fehler
* Strukturen passen nicht immer

Deshalb heute:

* Nur Klassen

Aktuell:

Entwicklungsbasis.Ausgeben(string text)

* Diese Methode soll den Text „normal“ für den Benutzer schreiben

Wie heißt gezielte Fehlersuche in einer Anwendung beim Programmieren?

* Debuggen

Wir wollen eine weitere Ausgeben Methode, die den Text nur für Entwicklung schreibt (zum Debuggen)

Neu seit .Net:

* Derselbe Methodenname kann öfter benutzt werden
* War vor .Net nicht erlaubt
* Ist heute möglich, wenn sich die Schnittstelle ändert
* Technik der „Überladung“

Wunsch (der Analyse):

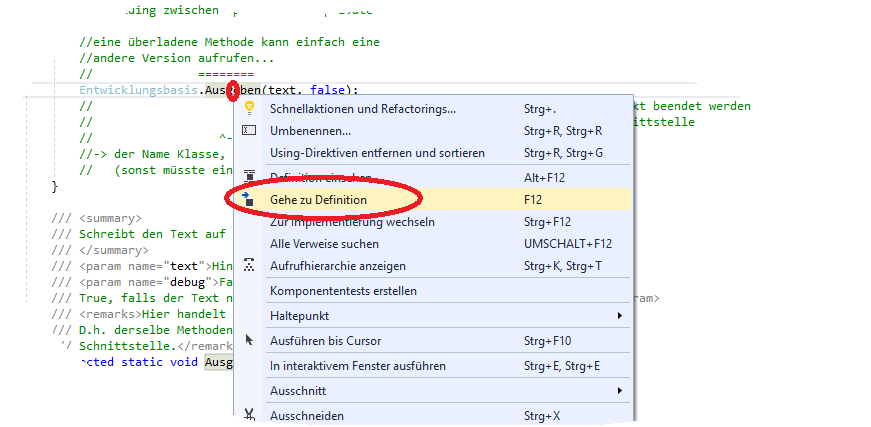
* Die Ausgeben() Methoden sollen OHNE Objektinstanz direkt aus der Klasse („Hausplan genehmigen“) genutzt werden können
* „static“ Modifizierer

Hinweis: 99,9 % der Methoden im .Net sind NICHT statisch, d.h. es wird ein Objekt benötigt zum Benutzen. Manchmal ist eine Methode „statisch“, kann also ohne Objekt direkt über die Klasse benutzt werden. (Ausnahme)

Fragen:

* Wofür steht „Entwicklungsbasis.Ausgeben(…)“?
  + Hier wird ein Programm, eine Methode aufgerufen
  + Feiner Unterschied zwischen „Anwendung“ und „Programm“ beim Trainer. Die Anwendung ist alles. Fertig am 22. November  
    „Programm“ entspricht einer „Methode“
* Warum „Entwicklungsbasis“. Das funktioniert ja mit „Ausgeben(…)“ alleine auch?
  + Aber, dann ist nicht ersichtlich, ob „Ausgeben“ einem Objekt gehört oder einer Klasse (statisch).
  + Entwicklungbasis ist eine Klasse
  + Man kann nur direkt aus einer Klasse eine Methode starten, wenn diese statisch ist
  + Außerdem: Konvention  
      
    Die Lesbarkeit ist der Kürze vorzuziehen

Oft benutzt:



* Visual Studio kann mit dem Befehl „Gehe zu Definition“ rasch den Cursor zur Stelle bringen, wo ein Ausdruck „deklariert“ ist.

Mit „Entwicklungsbasis.Ausgeben()“ sind die Vorbereitungsarbeiten abgeschlossen (im Grunde lauter Ausnahmen) und wir können uns den eigentlichen Aufgaben widmen:

Zurück zum Main:

* Hier soll ZeigeSequenz() aufgerufen werden

Weil ZeigeSequenz() nicht statisch ist,   
wird ein Objekt benötigt (99,9 % der Anwendungsfälle)

Standardablauf zum Benutzen einer Klasse als Objekt:

1. Speicher für das Objekt reservieren
2. Die Klasse als Objekt initialisieren
3. Das Objekt benutzen
4. Den Speicher für die Freigabe kennzeichnen

Neu:

* Schlüsselwort für „keine Speicheradresse“  
    
  null

Zusätzliches Schlüsselwort zum „Hausbauen“,  
zum Initialisieren einer Klasse als Objekt

* new

Das Initialisieren ist sehr zeitaufwändig und soll wirklich nur  
dann durchgeführt werden, wenn Objekt wirklich benötigt wird.

Microsoft sagt:

* Eine Anwendung muss so schnell als möglich starten
* Damit das gewährleistet ist, muss auf „new“ möglichst  
  verzichtet werden am Anfang.
* Objekte erst dann manchen, wenn’s der Benutzer anfordert

WICHTIG:

* Mit „new“ wird Speicher benötigt
* System.GC gibt den Speicher wieder frei,  
  wenn die Adresse eines Objekt nirgendwo mehr  
  in Verwendung ist.
* Deshalb: Objektvariablen auf „null“ setzen, wenn  
  diese nicht mehr benötigt werden.

Fragen:

* Wenn das Initialisieren von Klassen so aufwändig ist, warum macht man nicht alles statisch?
  + Z. B. Kundendatenblatt – Klasse
  + Würde jetzt „Name“ in der Klasse „statisch“ sein, gäbe es diesen GENAU EINMAL
  + D.h. statische Mitglieder sind die absolute Ausnahme
* Kann eine Klasse selber ein Objekt erstellen, warum erstellen wir das Objekt „außerhalb“?
  + Der gängige Weg
  + Es gibt aber auch die andere Lösung
  + Z. B. später sollen die Benutzer passend zur aktuellen Stunde begrüßt werden. Für die aktuelle Stunde  
      
    System.DateTime.Now.Hour  
      
    Die Hour Eigenschaft ist nicht statisch  
    Zum Glück aber die Eigenschaft Now. Das Interessante,  
    dass Now ein Objekt „DateTime“ zurückgibt

8. November 2018

Fragen:

* Bei diesen Aufrufen wie z. B. „Ausgeben(debug:true, text: "Hallo …")“  
  handelt es sich um benannte Parameter. Muss ich hier „debug:“ bzw. „text:“ benutzen oder kann ich auch „wurscht:“ schreiben?  
    
  NEIN, es muss ausdrücklich der Name des Parameters benutzt werden
* Frage, warum ändert „debug:true“ bzw.“debug:false“ derzeit nichts?

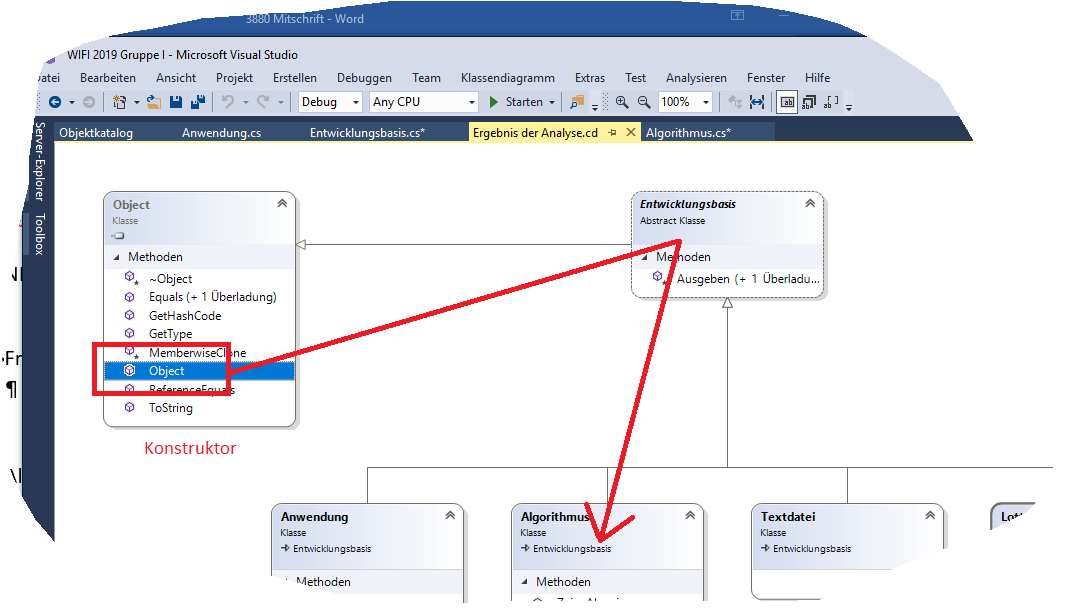
Weil wir nicht fertig sind

* Alles, was man sich wünscht, muss man auch selber machen!

Nächster Punkt:

* Wir wissen, dass mit dem „new“ Schlüsselwort aus einer Klasse ein Objekt erzeugt wird  
    
  Was wird dabei aufgerufen?  
    
  Der Konstruktor

Unser Code funktioniert deshalb, weil wir den Konstruktor …



… von System.Object geerbt haben.

Wunsch:

Wenn eine Algorithmus-Klasse initialisiert wird, soll „Ein Algorithmus Objekt lebt…“ ausgeben werden, wenn eine Textdatei initialisiert wird, „Ein Textdatei Objekt lebt…“ usw.

* Wir haben keine Lust, das drei bzw. noch öfter zu programmieren
* Deshalb  
    
  In die Entwicklungsbasis

Neuer Auftrag:

* In der Entwicklungsbasis einen Konstruktor hinzufügen, der den Text „Ein Objekt xyz lebt…“ (nur in der Testumgebung) ausgibt.

In BASIS heißt der Konstruktor „Sub New()“

In C# heißt der Konstruktor wie die Klasse

* Fast alle Klassen besitzen eigene Konstruktoren

Problem:

* Der Code der Entwicklungsbasis benötigt aber das eigentliche Objekt, bei uns zuerst einmal den Algorithmus

Lösung:

* „Zauberwort“ auf das aktuelle Objekt (in BASIC „Me“)  
    
  In C# „this“

Manchmal praktisch, verhindert aber, dass eine Anwendung „lokalisiert“, d.h. in andere Sprachen übersetzt werden kann…

* Einsetzen von Werte in einen Text  
  Erst seit C# 2015 möglich:  
    
  $"Blabla {Wert} blabla"
* Kann in der Realität nicht benutzt werden, weil für den Benutzer Texte „deutsch“, „englisch“, … ausgegeben werden müssen

Wie hat alles begonnen?

* IBM hat 1979 einen Prozessor, den 8088 auf den Markt gebracht
* Für einen Prozessor wird ein Betriebssystem benötigt
* IBM stellte Bill Gates ein, der für den 8088 (ein 8 Bit Prozessor) ein Betriebssystem entwickelte (=> PC-DOS)  
  PC-DOS, eine Textoberfläche, wurde für die max. 64 KB Speicher des Prozessor auf 640 KB ausgelegt
* Die Hardware Entwicklung hat die 640 KB rasch gesprengt
* IBM -> neues Betriebssystem. Weil mittlerweile Grafikkarten erfunden wurden, mit einer grafischen Oberfläche  
    
  OS/2  
    
  Stark vertreten: Bill Gates  
  Die beiden, IBM und Bill Gates, haben sich zerstritten.  
  Warum: Weil unter OS/2 die alte Software nicht mehr lauffähig war.  
    
  IBM-> dann verschwinde. Aber einen Teil darfst behalten:  
    
  Bill Gates hat Microsoft gegründet:  
  PC-DOS wurde als MS-DOS verkauft  
  Die grafische Oberfläche von OS/2 wurde als grafischer Zusatz zu   
  MS-DOS verkauft (um ca. 1984)  
    
  Devise von Gates: Fall’s etwas etwas Neues von uns gibt, muss das „Alte“ trotzdem laufen  
    
  Windows war kein Betriebssytem. Betriebssystem war MS-DOS bis   
  1995: Windows 95  
    
  bis Windows XP wurde als Klassenbibliothek die MFC benutzt. Die „Microsoft Foundation Class“. Problem: War nur mit C++ benutzbar  
    
  Mitte 1990iger hat Microsoft g’sagt, wir benötigten eine neue Klassenbibliothek:  
  + .Net Framework
  + Dieses Framework soll von mehreren Sprachen genutzt werden können. Und gleich ein moderneres C++: C#
  + Wichtig: C/C++ nicht schlechtmachen. Das ist IMMER NOCH DIE SPRACHE DER SYSTEMENTWICKLUNG

Zurück zum Thema:

* Zauberwort auf das aktuelle Objekt:  
    
  this
* Möchte man über das Objekt etwas in Erfahrung bringen, muss die .Net Technologie der „Reflection“ benutzt werden.  
    
  Einstiegsmethode von System.Object geerbt: GetType()

Hinweis:

* C# unterstützt  
    
  Escape Sequenzen  
    
  Diese werden benötigt, wenn ein Zeichen, z. B. das Anführungszeichen im Text benötigt wird.  
    
  z. B.  
  Eine Microsoft Textzeile, damit man in die nächste Zeile kommt, wird mit ZWEI Zeichen (im Gegensatz zu Linux, nur eines) abgeschlossen  
    
  \r\n  
    
  oder das Anführungszeichen: \"

Wichtiges Thema im .Net:

* Die Lebensdauer von Objekten

Früher:

* War bestimmt. Mit „New“ beginnt das Lebens eines Objekts (und verbraucht Arbeitsspeicher. Die Programmierung MUSST das Objekt zerstören durch Aufruf vom Destruktor. Falls das vergessen wurde, wurde der Speicher immer weniger.

Deshalb seit .Net:

* Der Destruktor kann nicht mehr aufgerufen werden.  
  Er wird vom Gargabe Collector verwaltet.
* Wann macht der Garbage Collector das?  
    
  Wann er „Zeit hat“  
  Hoch komplex. Nicht manuell eingreifen
* Fazit: Die Lebensdauer von Objekten ist „unbestimmt“

AUF KEINEN FALL auf das Aufrufen der Dispose-Methode vergessen, wenn diese vorhanden ist. Weil hier räumt das Objekt zusammen, was der Garbage Collector nicht wissen kann.

z. B. Schrift (im Teil 1)

Das ganze .Net ruft ja Betriebssystem Funktionen auf (Windows API)

Nächster Auftrag:

* Entwicklungsbasis.Ausgeben(string, bool) ist noch nicht fertig
* Falls der debug-Parameter auf true steht, die Schriftfarbe auf „Grau“ wechseln

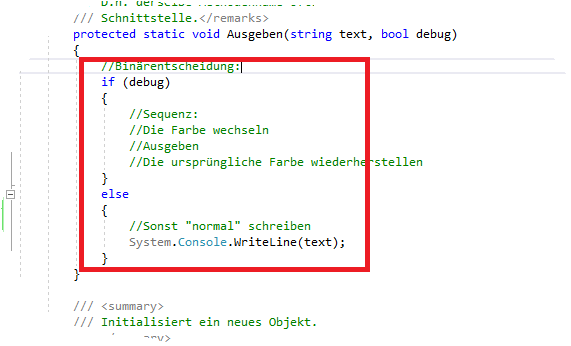
Lösung:

* Bisher nur eines nach dem anderen
* Jetzt  
    
  Eine Entscheidung, ein Entweder – oder (Binärentscheidung)  
    
  Alle Sprachen nutzen dazu…
* If – Schlüsselwort

Übrigens:

* Programmieren = Nachdenken
* Ergebnis: Algorithmus
* Bausteine des Algorithmus
  + Sequenz
  + Verzweigungen
  + Schleifen

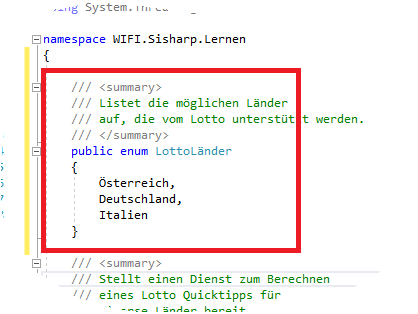
WEHE, es sagt irgendwer „if-Schleife“

Folgende Abbildung zeigt, wie die …  


… if – Anweisung in C# genutzt wird.

Praktisch in .Net

* Zusammengehörende Konstanten können als  
    
  Enumeration (Aufzählung, symbolisiert mit „gelben Zettel“)  
    
  abgebildet werden



Voi cool im Visual Studio seit .Net

* Projekt Konfigurationen
  + Für die Entwicklung (Debug)
  + Für die Kunden (Release)
* Man den Compiler für diese Debug bzw.  
  Release Variante mit „Präprozessordirektiven“  
  steuern. Diese beginnen alle mit dem Nummerzeichen (#)

Zum Schluss:

* Die Anwendung, die objektorientierte, wird größer und größer.
* Die Gefahr den Überblick zu verlieren steigt.
* Deshalb  
    
  Debug-Hinweise, die nur beim Entwicklung angezeigt werden

Gezieltes Testen einer Anwendung

* DEBUGGEN

Wichtige Tastenkombinationen:

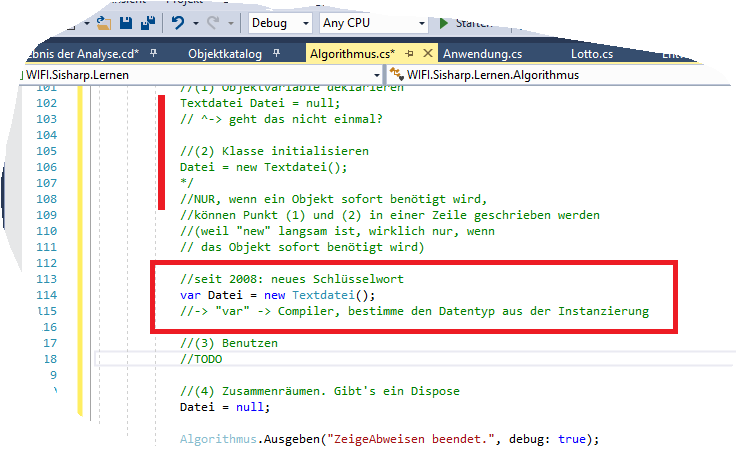
„Normal laufen lassen“ Funktionstaste F5

„Schritt für Schritt“ Funktionstaste F11

„Unterprogramm als ein Schritt“ Funktionstaste F10

13. Nov. 2018

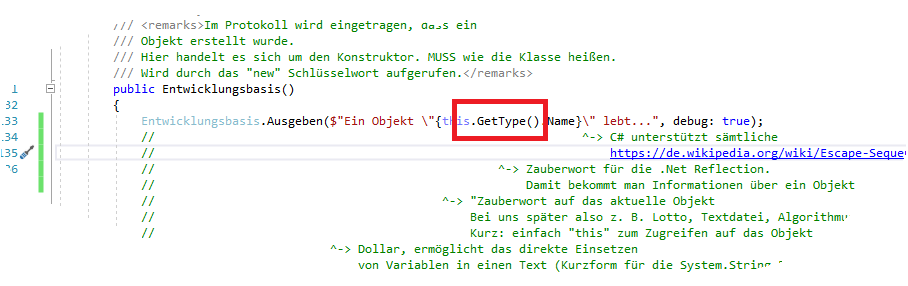
Seit 2008 kann sich der Compiler den Datentyp selber ermitteln…



* Schlüsselwort „var“

Wichtig:

* In einer Basisklasse weiß man nicht, wer einen benutzt
* Lösung:  
    
  this  
    
  als Schlüsselwort auf das aktuelle Objekt



* Benötigt man noch mehr Information über das Objekt,  
  mit der von Object geerbten Methode GetType() die Beschreibung des Objekts holen. Das ist die .Net Reflection Technologie.

Nächste Baustelle:

* Damit wir weitermachen können, muss die Entwicklungsbasis den Zufallsgenerator besitzen, damit wir würfeln können.

Überhaupt nicht leicht, mit dem Computer zu würfeln

* Wie heißt das C# Schlüsselwort (zum Würfeln)?  
    
  GIBT’s ned
* Problem
  + Entweder machen wir das selber  
      
    Lehmer: Lineare Kongruenzmethode  
    z. B. <http://home.eduhi.at/teacher/alindner/Sites/skripten/zufzahlen.pdf> (selber gesucht)

Restklassen-Division

5 / 2 -> 2,5

5 mod 2 -> 1

6 mod 2 -> 0

C# Operator für die Restklasse: % (Prozentzeichen)

* + Oder:  
      
    Wir hoffen auf .Net  
      
    System.Random

Wichtig:

* Grundprinzip:  
    
  Es gibt ihn nicht. Der Zufall ist über die Unendlichkeit gleichverteilt
* Deshalb darf eine Anwendung NUR GENAU EIN OBJEKT System.Random besitzen!
  + Wie kann sichergestellt werden, dass etwas nur genau einmal existiert?
    - Static

Wie wird eine Eigenschaft mit C# realisiert?

1. Substantiv (.Net Konvention)
2. Jede Eigenschaft hat einen Datentyp  
   (kein void)
3. Kennzeichen einer Methode: Schnittstelle mit den Klammern  
   KEINE Klammern bei Eigenschaften

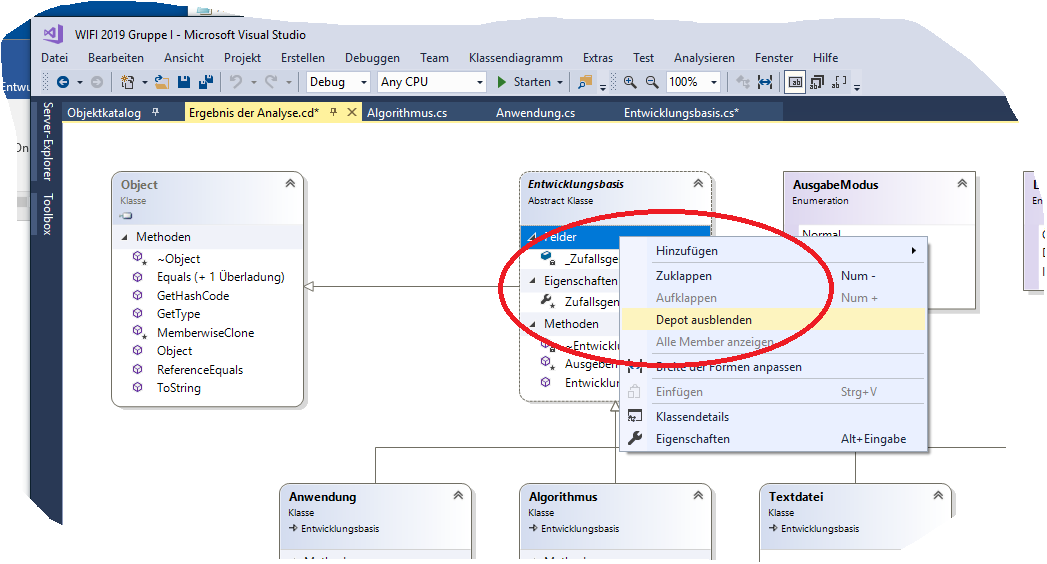
Problem beim Zufallsgenerator:

* Wir wissen nicht, wie oft die Eigenschaft abgerufen wird  
  Es muss nur jedes Mal funktionieren und immer dasselbe Random‑Objekt liefernt
* Deshalb kann die Eigenschaft sich den Wert nicht jedes Mal neu berechnen
* Wir müssen uns den Wert außerhalb der Eigenschaft merken
  + Die Eigenschaft ist Teil einer Klasse
  + Außerhalb der Eigenschaft: Klassenebene
  + Daten auf Klassenebene werden als „Felder“ bezeichnet

Eine Eigenschaft hat zwei untergeordnete Blöcke:

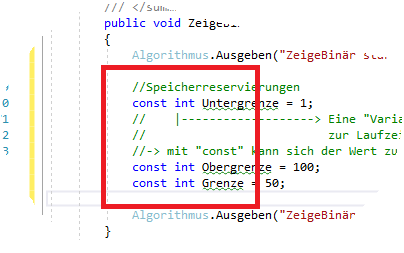
* Den „Getter“. Dieser Block läuft, wenn der Wert der Eigenschaft geholt wird
* Den „Setter“. Dieser Block läuft, wenn der Wert der Eigenschaft festgelegt wird. Der „Setter“ kann entfallen, wenn die Eigenschaft schreibgeschützt sein soll.

Weil die Felder …



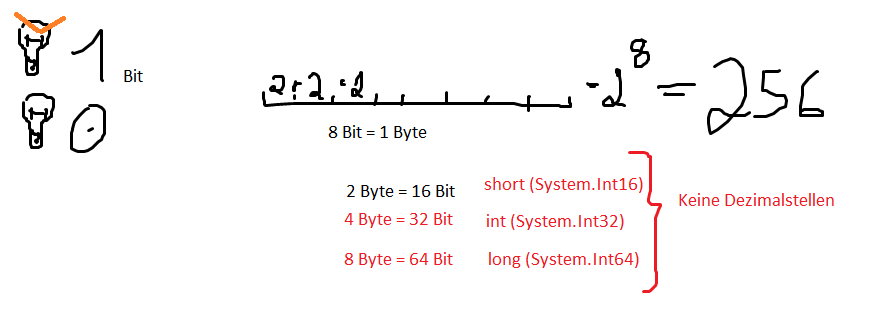
… IMMER PRIVAT sind, diese im Klassendiagramm „ausblenden“.

Speicherreservierung:



* Werden Daten benötigt, die zur Laufzeit nicht änderbar sind, diese als „Konstanten“ deklarieren

Wie stellt der Computer Ganzzahlen dar?



* Datentypen lernen   
  <https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/types>

Frage:

* Wann eine Methode, wann eine Eigenschaft?
  + Eine Methode arbeitet und gibt entweder nix   
    oder einen Wert zurück
    - Kennzeichen: Die Schnittstelle mit runden Klammern
    - Muss im Namen ein Verb enthalten
  + Eine Eigenschaft „beschreibt“
    - Kennzeichen: Keine runden Klammern
    - Muss ein Substantiv sein
  + Oft ist nicht klar, ob etwas mit einer Eigenschaft oder  
    mit einer Methode gelöst wird
    - Das bestimmt die Analyse
    - „Arbeitsgruppe“
    - Faustregel:
      * Dauert etwas länger, bis das Ergebnis bekannt ist:  
        Methode
      * Steht etwas „sofort“ bereit:  
        Eigenschaft

Warum erstellen wir den Zufallsgenerator nicht dort, wo er benötigt wird? Warum machen wir eine Eigenschaft?

* Grundsätzlich:  
    
  Dadurch wird wiederkehrende Arbeit erleichtert.  
  Das Benutzen von this.Eigenschaft ist einfacher als jedes Mal  
  var Objekt = new Klasse(); zu benutzen
* Bei Zufallsgenerator im Speziellen: Ein System.Random Objekt DARFS NUR GENAU EINMAL in der gesamten Anwendung (wegen der Gleichverteilung)

Zum ZeigeFall()

* Hier soll ein Begrüßungstext ermittelt werden

Im ZeigeZählen()

* Hier wird ermittle Begrüßungtext wieder benötigt

Deshalb:

* Eigene Methode ErmittleBegrüßung(int stunde)
  + Kein „void“
  + Eine Funktionsmethode, die einen Wert zurückgibt

15.11.2018

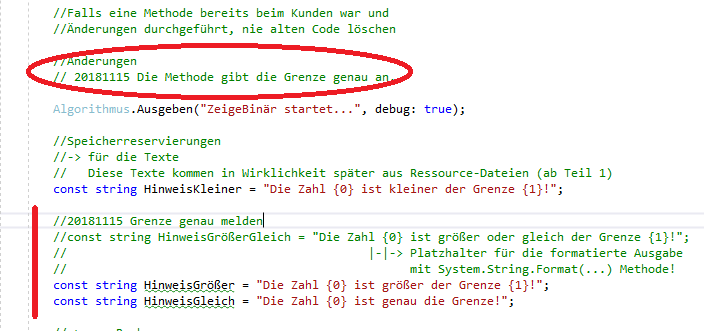
Frage:

* ***Wie behalte ich die Kontrolle über ein Projekt?***
  + Grundsätzlich  
      
    Das Ganze ist objektorientiert  
    So, wie beim Auto
    - Wenn das nicht anspringt,  
      kann sein, dass die Batterie kaputt ist
    - Beim Kaufen einer neuen Batterie,  
      fragt man sich nicht, wie die Batterie  
      gebaut wird
    - In objektorientierten Programmierung  
      ist das dasselbe, d.h. wenn ein Objekt (Klasse)  
      einmal funktioniert, weiß man nur mehr,  
      dass es gibt
  + Erster Schritt:  
      
    Damit man die Kontrolle nicht verliert,  
    das Klassendiagramm ansehen.
  + Eine Anwendung ist ein Projekt
    - Analyse
    - Ergebnis:   
      Klassendiagramm
    - Man macht nie eine komplette Anwendung
    - Nur einzelne Aufgaben:  
      Nämlich eine Methode zu implementieren
      * Was wollte die Analyse bei der Methode?
      * Algorithmus (Programmieren)
        + Sequenz
        + Verzweigungen
        + Schleifen
      * Tippen
        + Aktuell: Binärentscheidung – if

Frage

* Wie kann ich den die Eigenschaft testen?
* Wo wird’s benötigt?
  + Im ZeigeDurchlaufen() vom Algorithmus

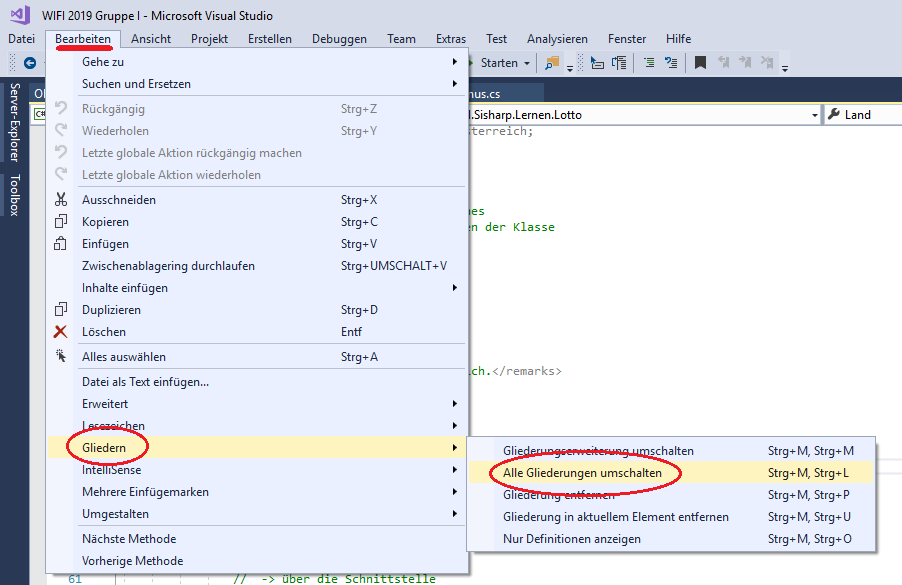
Tipp beim „Warten“ von Anwendungen



* Nie alten Code löschen
* Alten Code auskommentieren
* Mitprotokollieren, wann wer was geändert hat

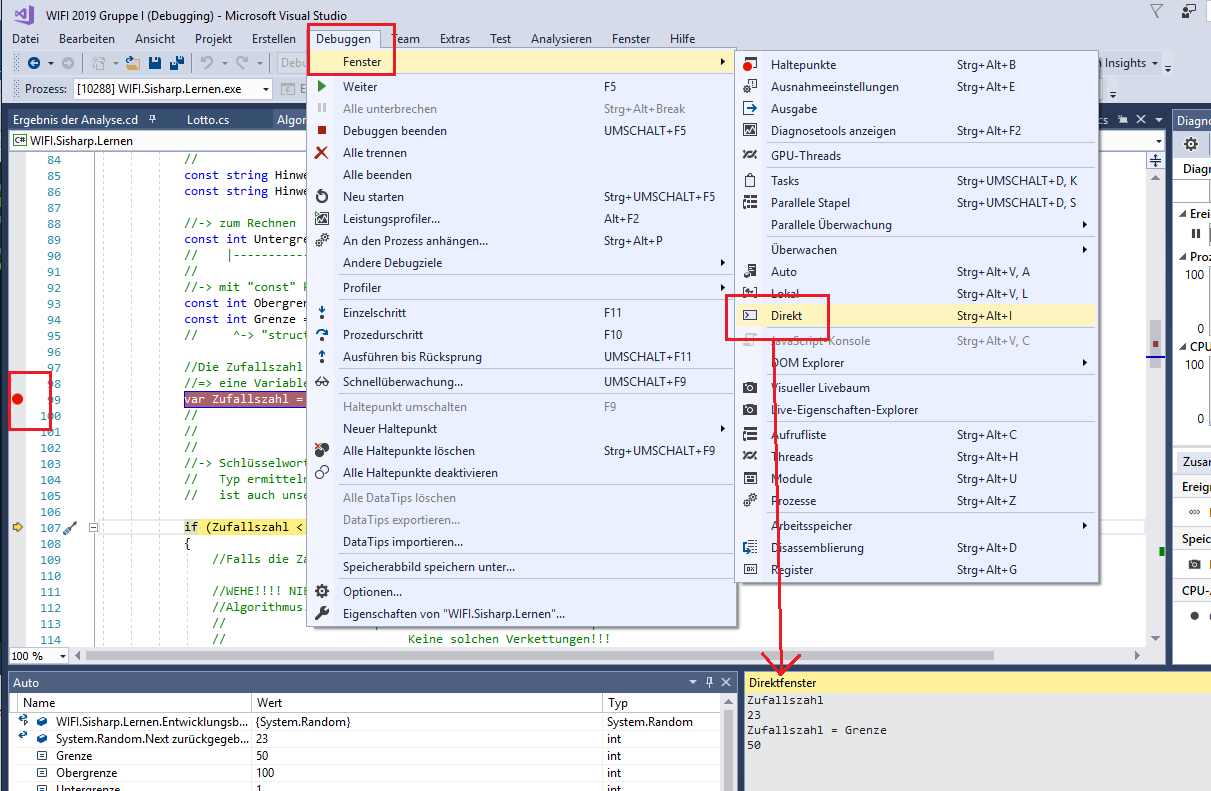
Frage:

* Können die Kommentare ausgeblendet werden?



Jein -> nur die Xml Kommentare befinden sich in einer Gliederung. Diese können mit Strg+M, Strg+L umgeschaltet werden.

* Wie kann ich beim ZeigeBinär auf Gleich testen?

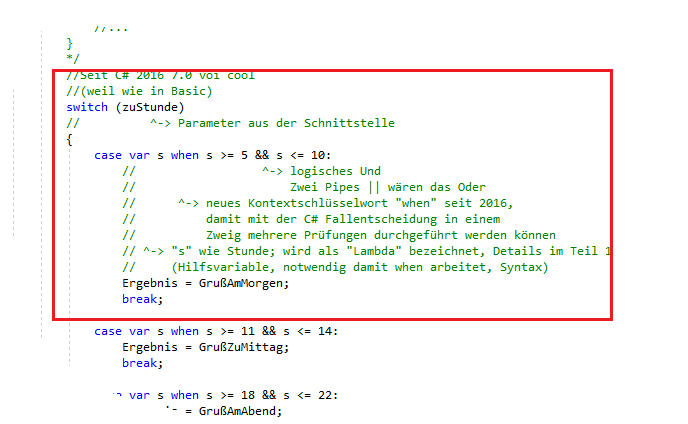


* Nicht im Programmtext „harte“ Zuweisungen tippen
* Warum?
* Weil sich dadurch unter Umständen Programmfehler einschleichen
* Richtig:
  + Haltepunkte setzen
  + Im Direktbereich können Variablen-Inhalte überschrieben werden

Zurück zum ErmittleBegrüßung(…)

* Wir müssen 4 Fälle unterscheiden
* Möglich
  + Drei Binärentscheidungen
  + Fad
* Eleganter
  + Es wird ja nur „zuStunde“   
    zugewiesen
  + Fallentscheidung  
      
    BASIC: Select Case

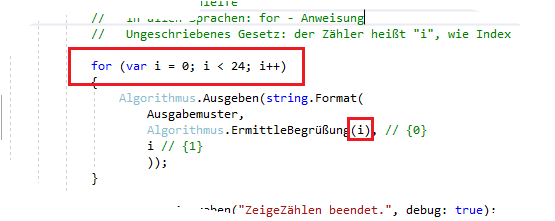
C#: switch



Zum ZeigeZählen

* Soll ErmittleBegrüßung für jede Stunde testen

Lösung: Die Zählschleife „for"



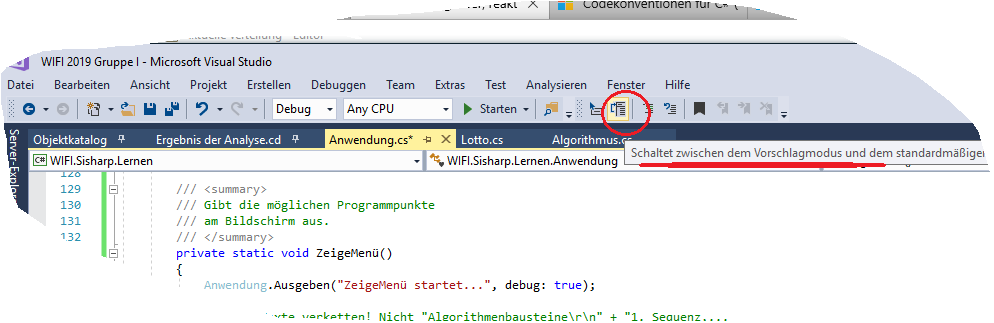
Nächste Baustelle:

* Die Ausgabe wird länger und länger
* Wir können die Beispiele nicht sequentiell durchlaufen lassen
* Wir wollen den Benutzern die Möglichkeit   
  geben, zu entscheiden, welches Beispiel   
  gezeigt wird

Main zusammenräumen

* Eine zusätzliche Methode „ZeigeMenü()“ in der Anwendung

Frage:

* Mein Visual Studio markiert die Mitglieder beim Tippen eines Objekts nicht mehr in der Liste. Warum?
  + 

20. November 2018

Frage:

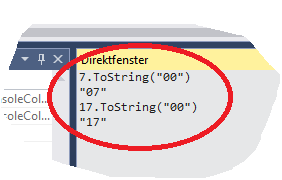
* Müssen die Namen der Parameter einer Methode und die Namen der Variablen, die an diese Parameter übergeben werden, gleich lauten?
  + Nein
  + In fast allen Fällen heißt das anders
  + Der Datentyp muss identisch sein
* Wann darf ich eine Methode statisch machen?
  + In den seltensten Fällen (fast nie)
  + Ausnahmen
    - Statisch heißt, dass das Mitglieder der Klasse (z. B. Kundenstammblatt) und nicht dem Objekt (z. B. ErsterKunde, ZweiterKunde, ….) gehört
    - Möglich:  
        
      EuroKursATS = 13,7603  
        
      Wenn 100 % sicher ist, dass etwas nur GENAU EINMAL existiert, dann „static“
    - Das überlegt die Analyse.

---

Hinweis:

* Auch in C# kann sich eine Methode „selber“ aufrufen
* So etwas heißt „Rekursion“
  + Oft extrem kurze Lösungen
  + Schwer zu debuggen
* Bei einer irrtümlichen Rekursion:  
  „Stapel-Überlauf“ Absturz

Frage

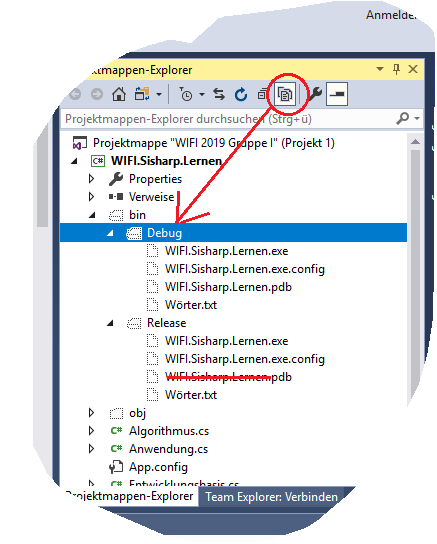
* In Excel kann ich Zahlen formatieren…  
  Wie formatiere ich Zahlen in .Net?  
    
  + .ToString() – Überladung  
    .ToString("00")  
     
  + Es gelten mehr oder weniger dieselben Formatierungscode, die auch Excel kennt, allerding englisch

Damit der Benutzer auswählen kann, welches Beispiel er sehen möchte…

* Die Durchlaufescheife  
  (Fußgesteuerte Bedingungsschleife)  
    
  C# benutzt dazu die
* do – Anweisung



Praktisch:



Im Studio können beliebige Dateien in einen Projekt-Ordner hinzugefügt werden. Diese werden vom Studio beim Kompilieren in den Zielordner kopiert.

Wichtig:

* Der Kunde bekommt die Release Kompilierung
* IMMER OHNE  
  der „PDB“ Datei.  
    
  Das ist die Debug-Datei mit den Namen der benutzten Variablen.  
  Falls die Datei beim Kunden ist, kann dieser (auch wenn’s verboten ist) die Anwendung „dekompilieren“ und sieht sogar, wie wir die Variablen genannt haben

Problem:

* Wie finden wir das Anwendungsverzeichnis?
  + C# hat nix
  + .Net hat nix
  + Selber machen
* Entwicklungsbasis.Anwendungsverzeichnis

Hauptthema in der Programmierung:

* Eine unbekannte Anzahl an Daten
* Eine unbekannte Anzahl an Daten  
  erfordert eine unbekannte Anzahl  
  an Variablen
* Nicht durchführbar  
  Wir können nicht …  
  string ErsteZeile = string.Empty;  
  string ZweiteZeile = string.Empty;  
  string DritteZeile = string.Empty;  
  …???  
    
  implementieren.
* Lösung für solche Probleme  
    
  EINE Variable für mehrere Werte  
  So etwas heißt…  
    
  **Datenfeld, Array oder Liste (Collection)**  
  Drei Begriffe für dasselbe.

C# kennt nur „statische“ Datenfelder, die sich   
im Umfang nicht verändern können.

Wir benötigen für unseren Fall aber ein…

* Dynamisches Datenfeld

… das zur Laufzeit im Umfang erweitert werden kann.

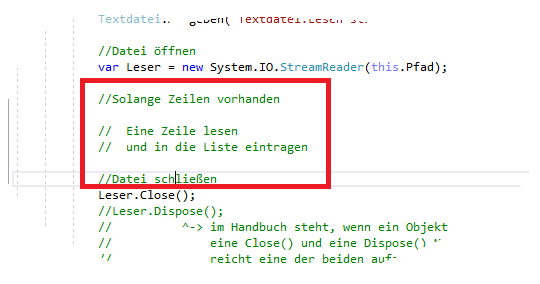
* .Net System.Collections…

22. November 2018

* Zurück zur Textdatei:

Weil die Textdatei eine unbekannte Anzahl an Zeilen hat, benötigten wir ein Datenfeld, das sich dynamisch vergrößern kann

* System.Collections.ArrayList
* Dazu eine Eigenschaft „Zeilen“
* Wie werden folgende Problemfälle getippt?

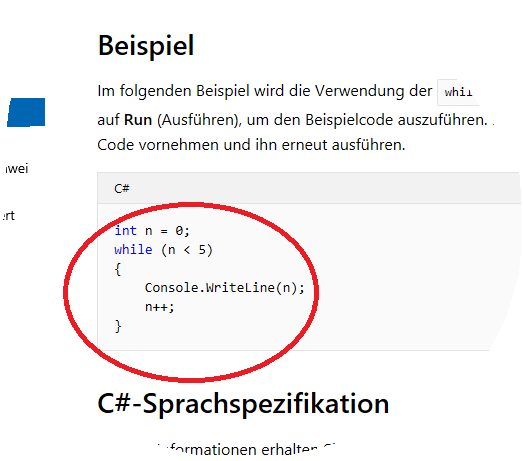


* Eine Schleife, wo nicht bekannt ist, wie oft der Inhalt benötigt wird, unter Umständen nie
* „kopfgesteuerte Bedingungsschleife“ (Abweiseschleife)

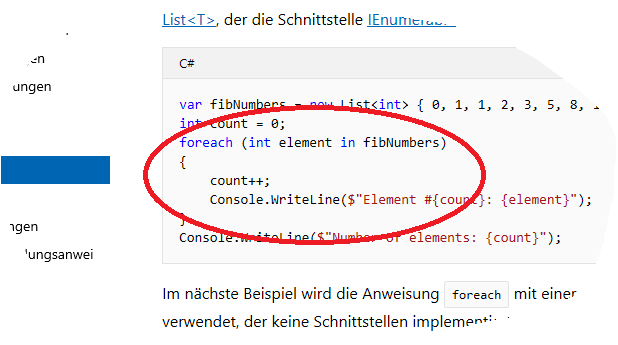
Alle Sprachen nutzen dazu das …

* while

… Schlüsselwort.

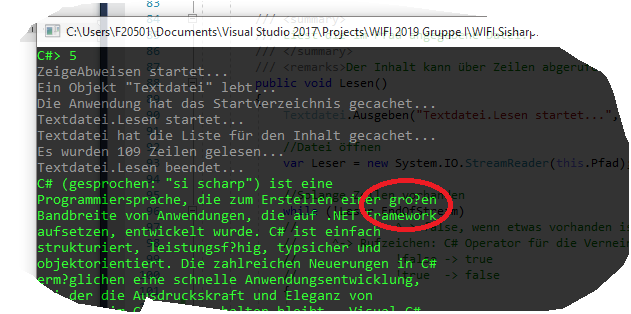


Wird jedes Element einer Liste benötigt, …



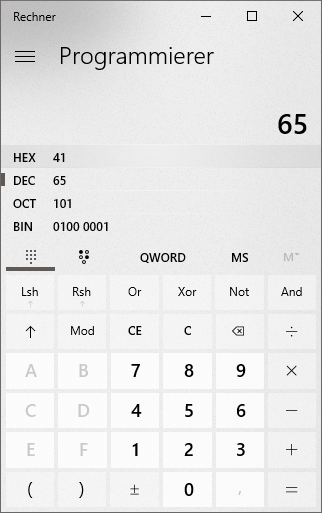
… kann die „Spezialzählschleife“ „foreach“ anstelle von „for“ benutzt werden.

Problem:

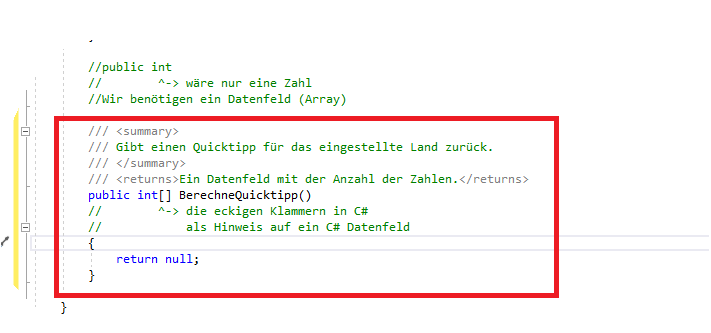


* Unser StreamReader interpretiert die Datei falsch (falscher Zeichensatz)

Wie wird Text im Computer dargestellt?

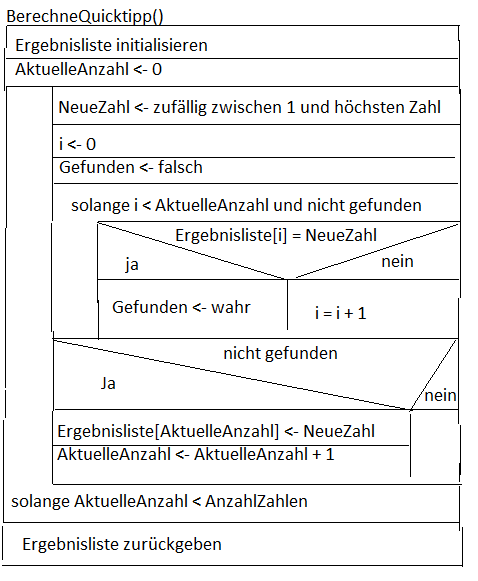
* Mit Zahlen, z. B. Dezimal 65 ist Binär   
  
* Irgendwer ist hergegangen und hat fixiert,   
  dass die Zahl 65 das große A ist
* Das sind die (Computer)Zeichensätze
  + ASCII (American Standard Code for Information Interchange)  
    7 Bit Zeichensatz, 128 Zeichen
  + Erweiterten ASCII für MS-DOS  
    8 Bit Zeichensatz, 256 Zeichen
  + ANSI – Zeichensatz (ab Windows 1 – XP)  
    8 Bit Zeichensatz, 256 Zeichen, die erste Hälfte ist identisch, die zweite ist anders zum erweiterten ASCII
  + HEUTE  
      
    UNICODE – Zeichensatz (seit Windows Vista)  
    (16 Bit Zeichensatz) ca. 65.536
* Mit System.Text.Encoding kann .Net jeden Zeichensatz darstellen

Zu unserer letzten Aufgabe:



* Ablauf zum Umsetzen einer Aufgabenstellung

1. Lösungsidee  
     
   Zufällig eine Zahl zwischen 1 und der höchsten Zahl berechnen  
   bis die Anzahl der Zahlen vorhanden ist.  
   Lotto ist aber ein „Ziehen ohne Zurücklegen“, d.h. dieselbe Zahl darf nur einmal vorkommen.
2. Bestandsaufnahme (Inventur)  
   Vorhanden ist:  
   -> der Zufallsgenerator aus der Entwicklungsbasis  
   -> Die Anzahl der Zahlen  
   -> Die höchste Zahl
3. Algorithmus (Nachdenken)  
   Darstellungsformen:
   1. Pseudocode  
      „Ein Aufsatz mit Sequenz, Verzweigung und Schleifen“
   2. Nassi und Shneiderman  
      „Struktogramm“



1. Tippen
2. Testen