Windows, mit Microsoft Konto automatisch anmelden

Ich habe ja kein Problem auch meinen Microsoft Account dazu zu nutzen um mich an meinen lokalen Rechner anzumelden.

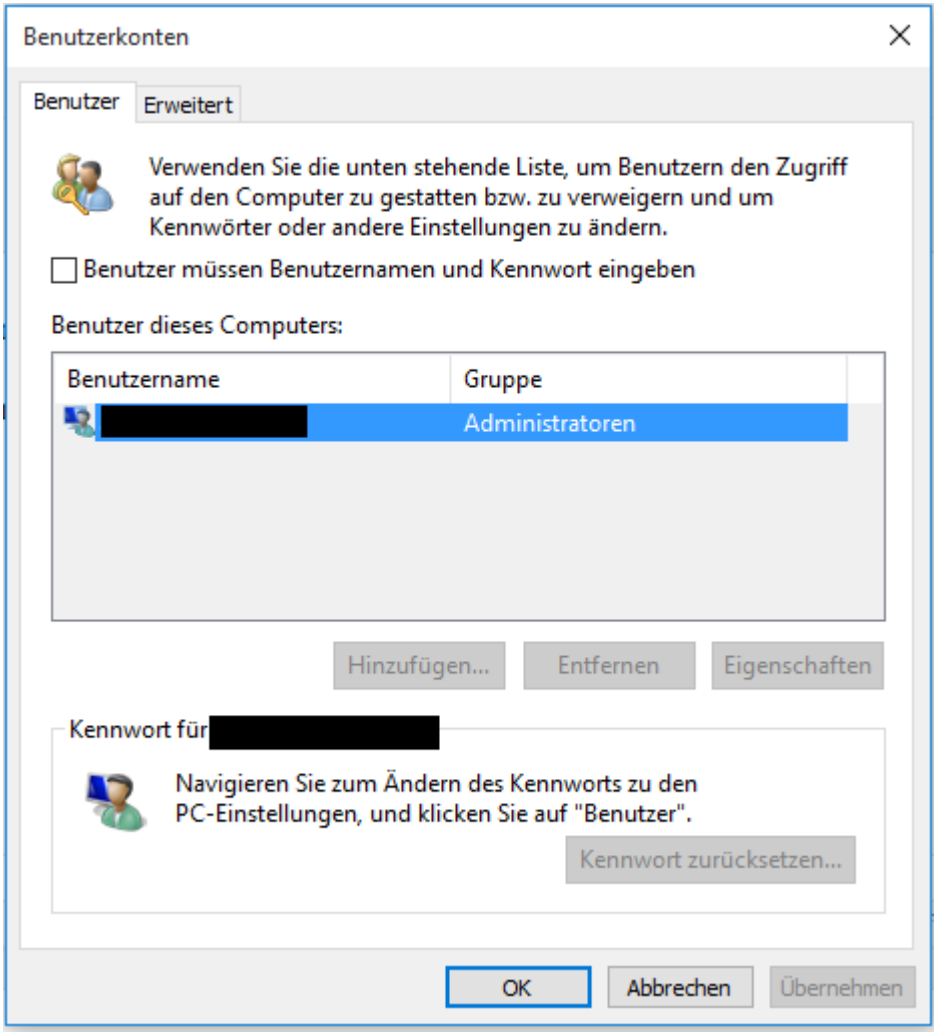
Allerdings nervt mich, dass ich nach jedem neustart das Passwort des Microsoft Accounts eingeben muss.

Dies geht auch einfacher:

1. Das lokale Konto evtl. unter Kontoeinstellungen (Systemsteurung) in Microsoft-Konto wechseln und dann

In die Suche oder unter Ausführen netplwiz eingeben und den Haken bei „Benutzer müssen Benutzernamen und Kennwort eingeben“ wegnehmen.

dann das letzte Mal das Passwort eingeben und Fertig



SQL Server Querys

Duplicate löschen, welche sich aus col1 und col2 zusammen setzen. Alternativ kann man natürlich auch nur mit col1 arbeiten
[crayon-677df06fa6c05804294393/]

Update Spalte, wenn Datum Heute > dann 1 sonst 0:
[crayon-677df06fa6c0c150072389/]
Zeilennummerierung erzeugen:
[crayon-677df06fa6c0e199583453/]

LINQ Querys

SQL:

IF EXISTS (SELECT 1 FROM tblUsers WHERE Username = „whatever“)
// Return 1, wenn Username vorhanen ist, 0, wenn eben nicht

LINQ:
[crayon-677df06fa70f3176499046/]

My SQL Querys

eine Sammlung netter Querys für MySQL:

ifnull(expression,“) – Wenn expression NULL ist, dann zählt der rechte Wert.

Concat(,a‘,123,(SELECT col from tab WHERE a = b)) – erstellt einen zusammengesetzten String aus allen teilen der Funktion

Eigenen Event erstellen

Ein Event oder deutsch Ereignis ist immer an einen Delegatentypen gebunden, welcher wiederum eine bestimmte Signatur von (Rückgabewert + Parameter).

Ungefähr, stellt es dieses Muster da:

Zuerst brauchen wir einen delegaten. Jetzt stellt sich die Frage, die Methoden, auf die der Delegat später verweist, wie soll die aussehen? soll sie einen Rückgabewert haben? Soll irgend ein Wert an die Methode übergeben werden? Ja? Dann brauchen wir Parameter. Um das noch ein wenig verständlicher zu machen, wenn wir eine for-Schleife haben und wollen, dass das Event nach jedem Schleifendurchgang immer abgefeuert wird, und uns den Wert liefert. Dann würden wir einen Delegaten mit einem Parameter vom Typ int z.B. machen.
[crayon-677df06fa72d6136238793/]

Dann braucht man das Event an sich. Dieser muss vom Typ des Delegaten sein.
[crayon-677df06fa72da310926165/]

Jetzt muss gesagt werden, wann das Event abgefeuert werden muss. Und, wichtig, wir haben dem Delegaten einen Parameter (int a) angegeben. Dieser Parameter muss nun durch das Event gefüllt werden. Das machen wir, indem wir
[crayon-677df06fa72db754988203/]
Jetzt wird bei jedem Schleifendurchgang das Event abgefeuert.

Ok, das war unsere Quell klasse.

Jetzt erstellen wir ein Objekt dieser Klasse und weisen dem Event eine Methode zu, die ausgeführt werden soll, sobald das Event abgefeuert wird. Die Methode muss die selbe Signatur haben wie unser Delegat. Zum Schluss müssen wir die Methode in der Quell klasse ausführen und das Event wird abgefeuert und führt ebenso die Methode mit dem Parameter aus, die wir in der Ziel Methode eingegeben haben.
[crayon-677df06fa72dc594354054/]

[crayon-677df06fa72de087769516/]

Download Beispielprojekt Eigener Event

TSQL Mod – Eine DLL für schnelle SQL Server arbeiten

TSQLmod Download

Klasse db (Connection String)

benötigt in erster Linie die SQL Instanz. Nachdem das Objekt erfolgreich initialisiert wurde, wird auch gleichzeitig die Verbindung aufgemacht und die folgenden Methoden können genutzt werden.

LookUP(…)

gibt aus einem SQL Query den ersten Treffer der angegebenen Spalte als String wieder. Ideal um einen Wert aus der Datenbank auszulesen. Möglich ist es entweder die Spaltennummer oder den Spaltennamen anzugeben.

getRowList(…)

gibt eine List<string> oder generische List<T> von der angegebenen Spalte zurück. Man erhält quasi aus dem Select eine gewünschte Spalte

getRowStringBuilder(…)

ähnlich wie die getRowList(…) ist der Rückgabewert aber ein

Stringbuilder, in welchen alle Zeilen einer selektierten Spalte enthalten sind.

getDynamicList(…)

erfordert eine Klasse welche dieselben Datentypen und Bezeichnung hat wie das SQL Select. Als Rückgabe erhält man man eine List<meineKlasse>, welche 1:1 so viele Elemente und Spalten hat wie das Sql Query. Das ganze arbeitet nicht mit Reflektionen, sondern nach dem Prinzip von diesem genialen Autor: KLICK

Dazu jeweils ein Beispiel. Ausgehend vom folgenden Select:

```
SELECT TOP 20 [ContactTypeID]
      ,[Name]
      ,[ModifiedDate]
  FROM [AdventureWorks2014].[Person].[ContactType]
```

100 %

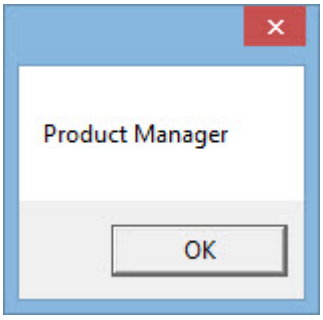
Ergebnisse | Meldungen

	ContactTypeID	Name	ModifiedDate
1	1	Accounting Manager	2008-04-30 00:00:00.000
2	2	Assistant Sales Agent	2008-04-30 00:00:00.000
3	3	Assistant Sales Representative	2008-04-30 00:00:00.000
4	4	Coordinator Foreign Markets	2008-04-30 00:00:00.000
5	5	Export Administrator	2008-04-30 00:00:00.000
6	6	International Marketing Manager	2008-04-30 00:00:00.000
7	7	Marketing Assistant	2008-04-30 00:00:00.000
8	8	Marketing Manager	2008-04-30 00:00:00.000
9	9	Marketing Representative	2008-04-30 00:00:00.000
10	10	Order Administrator	2008-04-30 00:00:00.000
11	11	Owner	2008-04-30 00:00:00.000
12	12	Owner/Marketing Assistant	2008-04-30 00:00:00.000
13	13	Product Manager	2008-04-30 00:00:00.000
14	14	Purchasing Agent	2008-04-30 00:00:00.000
15	15	Purchasing Manager	2008-04-30 00:00:00.000
16	16	Regional Account Representative	2008-04-30 00:00:00.000
17	17	Sales Agent	2008-04-30 00:00:00.000
18	18	Sales Associate	2008-04-30 00:00:00.000
19	19	Sales Manager	2008-04-30 00:00:00.000
20	20	Sales Representative	2008-04-30 00:00:00.000

einem erstellten Objekt der Klasse db:
[crayon-677df06fa7470849113391/]
und ein string mit folgendem select:
[crayon-677df06fa7474831608895/]

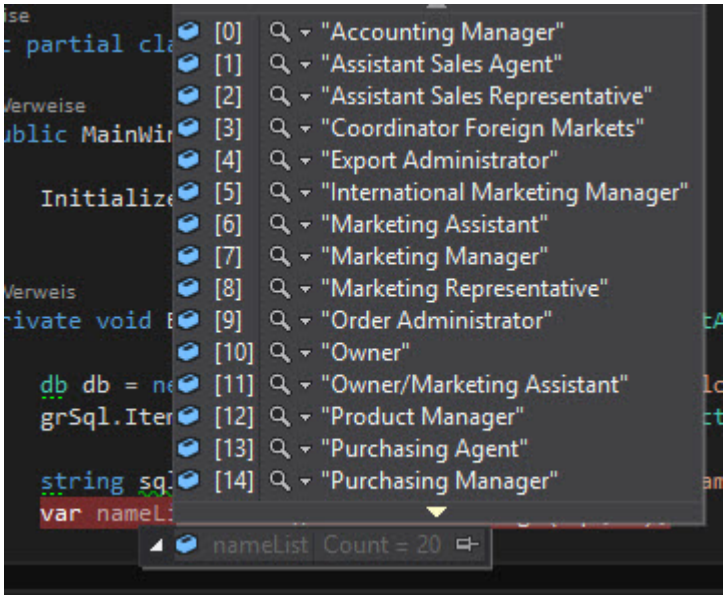
LookUp(…)

[crayon-677df06fa7476007423158/]

Antwort: 

getRowList(…)

[crayon-677df06fa7477296617021/]

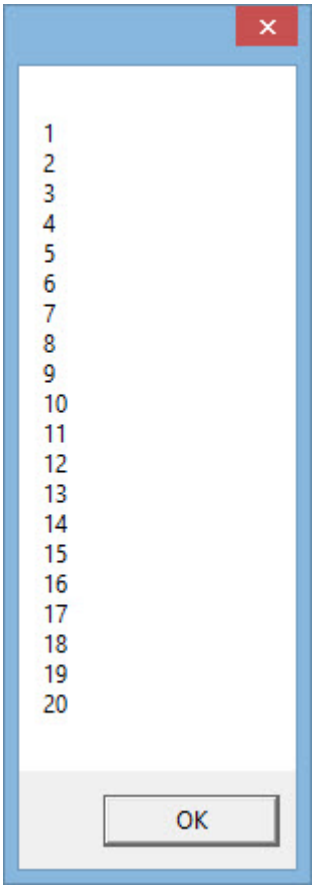


Antwort:

getRowStringBuilder(…)

[crayon-677df06fa7479511187667/]

Antwort:

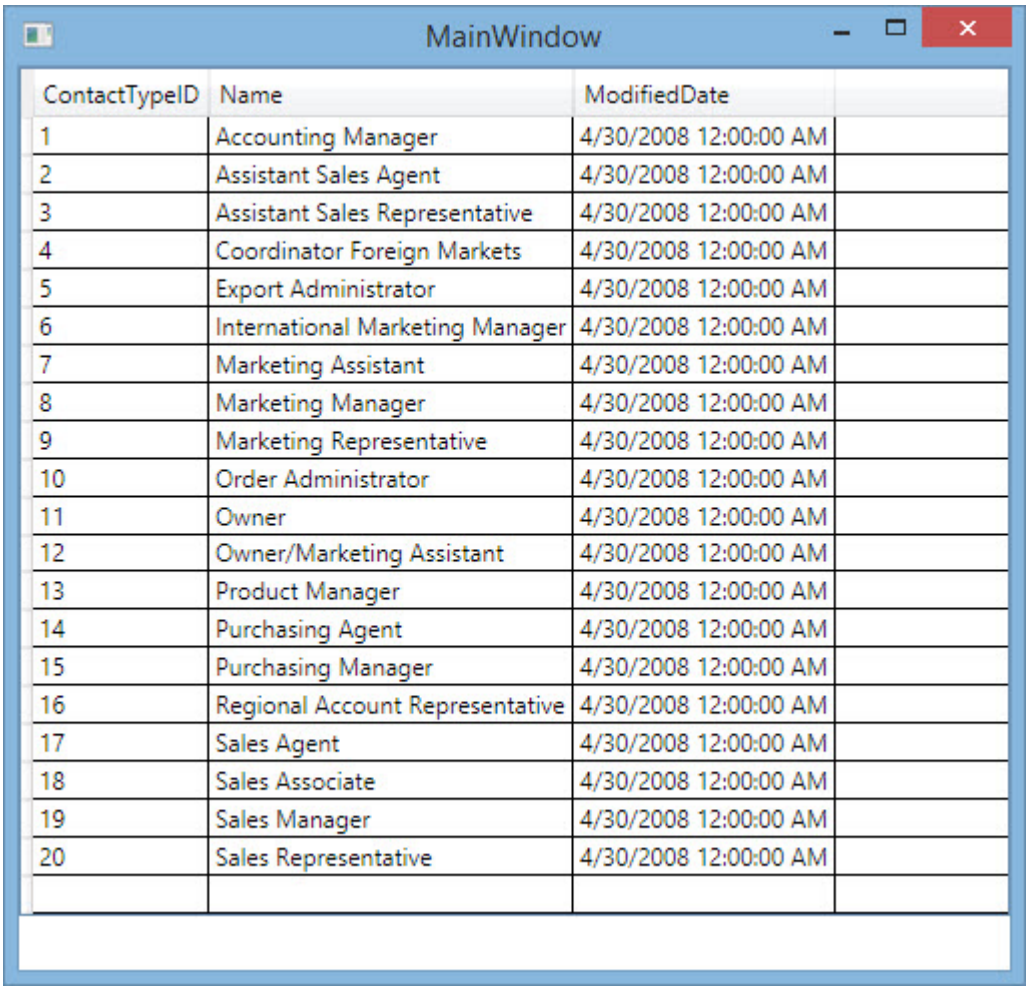


getDynamicList(…)

Wie oben bereits erwähnt, ist hierfür eine Klasse mit

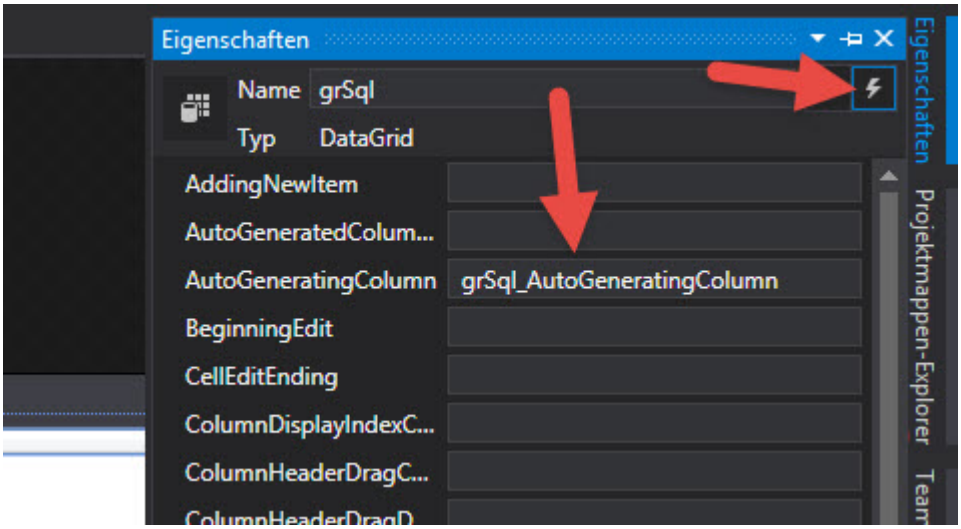
Propertys notwendig. Diese kann man ganz einfach auch mit den Methoden aus CreateClass – Klasse erstellen. Dazu weiter unten.
[crayon-677df06fa747a784659744/]
dann kann man so eine dynamische Liste ganz einfach erstellen:
[crayon-677df06fa747b804895441/]

diese Liste kann man nun z.B. einem Datagrid aus WPF zuordnen:
[crayon-677df06fa747c491343664/]
[crayon-677df06fa747e367522003/]
Das ganze sieht dann so aus:



ContactTypeID	Name	ModifiedDate
1	Accounting Manager	4/30/2008 12:00:00 AM
2	Assistant Sales Agent	4/30/2008 12:00:00 AM
3	Assistant Sales Representative	4/30/2008 12:00:00 AM
4	Coordinator Foreign Markets	4/30/2008 12:00:00 AM
5	Export Administrator	4/30/2008 12:00:00 AM
6	International Marketing Manager	4/30/2008 12:00:00 AM
7	Marketing Assistant	4/30/2008 12:00:00 AM
8	Marketing Manager	4/30/2008 12:00:00 AM
9	Marketing Representative	4/30/2008 12:00:00 AM
10	Order Administrator	4/30/2008 12:00:00 AM
11	Owner	4/30/2008 12:00:00 AM
12	Owner/Marketing Assistant	4/30/2008 12:00:00 AM
13	Product Manager	4/30/2008 12:00:00 AM
14	Purchasing Agent	4/30/2008 12:00:00 AM
15	Purchasing Manager	4/30/2008 12:00:00 AM
16	Regional Account Representative	4/30/2008 12:00:00 AM
17	Sales Agent	4/30/2008 12:00:00 AM
18	Sales Associate	4/30/2008 12:00:00 AM
19	Sales Manager	4/30/2008 12:00:00 AM
20	Sales Representative	4/30/2008 12:00:00 AM

wem das Datumsformat stört, der kann dem Ereignis AutogeneratingColumn aus dem Datagrid eine Änderung des Datumsformates durchführen:



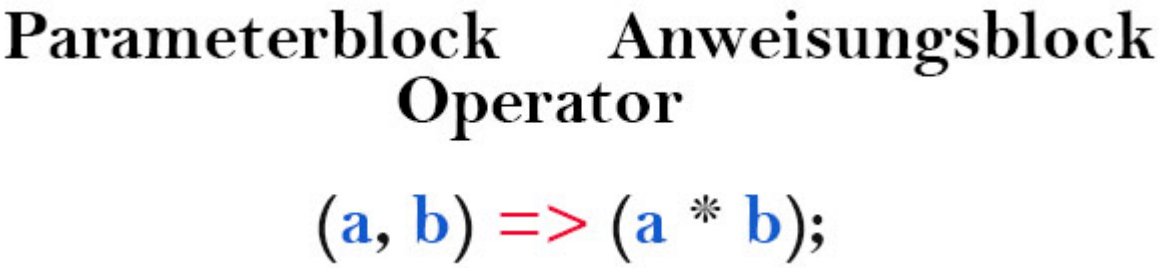
folgendes soll nun passieren, wenn das Ereignis eintrifft:
[crayon-677df06fa747f420286120/]
Nun sieht das ganze so aus:

ContactTypeID	Name	ModifiedDate
1	Accounting Manager	30.04.2008
2	Assistant Sales Agent	30.04.2008
3	Assistant Sales Representative	30.04.2008
4	Coordinator Foreign Markets	30.04.2008
5	Export Administrator	30.04.2008
6	International Marketing Manager	30.04.2008
7	Marketing Assistant	30.04.2008
8	Marketing Manager	30.04.2008
9	Marketing Representative	30.04.2008
10	Order Administrator	30.04.2008
11	Owner	30.04.2008
12	Owner/Marketing Assistant	30.04.2008
13	Product Manager	30.04.2008
14	Purchasing Agent	30.04.2008
15	Purchasing Manager	30.04.2008
16	Regional Account Representative	30.04.2008
17	Sales Agent	30.04.2008
18	Sales Associate	30.04.2008
19	Sales Manager	30.04.2008
20	Sales Representative	30.04.2008

Lambda Ausdrücke oder anonyme Methoden

Lambda Ausdrücke werden in Kombination mit Delegaten dazu verwendet um schnelle Anweisungen auszuführen und sich dabei jede Menge Code zu ersparen.

Ohne Lambda und anonymer Methode für ein Delegatenaufruf in etwa so aussehen:



Neue Eigenschaft vom Typ delegate erstellen. (Rückgabewert ist ein int, mit 2 in Parameter a und b)
[crayon-677df06fa768e860252899/]
Dann müssen wir eine Methode haben, die ebenfalls denselben Rückgabewert und Parameter besitzt (Anzahl und Typ sind entscheidend)
[crayon-677df06fa7692419888623/]
Nun könnte man ein Objekt vom den oben erstellten MyDelegate erstellen und dem die Methode addiere zuweisen:
[crayon-677df06fa7693399105076/]
Durch den Aufruf des Objektes dlg zeigt der Delegat auf die Methode addiere, führt diese aus und gibt den Rückgabewert zurück:
[crayon-677df06fa7694194421009/]

_____

Dies ist in der Tat etwas umständlich. Vor allem benötigt man immer ein Methode, die man dem Delegaten immer zuweisen muss.

schneller geht es mit den Lambda Ausdrücken.:
[crayon-677df06fa7695411280702/]
Die delegaten delegat1 bis delegat 5 tun immer dasselbe, nur wird der Code dadurch sehr kompakter und wenn man delegaten erstmal verstanden hat, wird es auch übersichtlicher.

Hat man einen Delegaten ohne Rückgabewert, würde der Lambdaausdruck mit Klammer auf- Klammer zu sein:
[crayon-677df06fa7696737413731/]

Wegen den => Operator werden die Lampdaausdrücke oft mit Linq verwechselt. Das eine hat mit dem anderen aber nichts zu tun

Vom Projekt bis Fertigstellung – Teil 1: Projektbeginn, Programmablaufplan (PAP)

Neben der eigentlichen Arbeit eines Entwicklers am Programmcode, sind vor allem für größere Projekte Diagramme, Notizen, und Pläne von sehr hoher Bedeutung und werden nur zu gerne unterschätzt. Vernachlässigt man dieses, verliert den Durchblick und am Ende die Motivation überhaupt weiter zu machen. Was ist also wichtig, wenn man ein Projekt plant?

1. Ziele setzen

Wichtig ist, dass man zunächst nicht alle gewünschten Features auf einmal versucht einzubauen, sondern dies bewusst trennt. Auf der einen Seite habe ich die Grundfunktionen bzw. das eigentliche Programm und auf der anderen Seite der Wichtigkeit nach sortierte Features, die ich danach implementieren werde. Beides schreibt man am besten auf ein Blatt Papier, trennt sie und legt sich den Basisfunktionszettel gut Sichtbar auf dem Schreibtisch.

2. Welche Technologien? Welche Themen?

Welche Programmiersprache und Datenbank zum Einsatz kommt ist eher die leichtere Frage, die geklärt werden muss. Wichtiger ist, dass man sich dann aber mit den Themen, die auf einen zukommen genau informiert. Möchte man einen Server programmieren, so schaut man sich die Protokolle genau an und lernt erstmal den theoretischen Teil.

3. Programmablaufplan (PAP)

Wieder mit Zettel und Stift bewaffnet kann man nun einen ungefähren Ablauf des Programms erstellen. Wichtig ist, dass man sich hier eben nur auf die Basisfunktionen beschränkt. Der Rest kommt, wenn alles fertig ist. Hier gibt es einen einfachen Einstieg in PAP -> http://www.mrknowing.com/2014/03/13/wie-erstelle-ich-ein-pap-programmablaufplan/. Vielleicht ist es nicht wichtig die einzelnen Formen genau zu kennen, doch ist es sehr Hilfreich, sich im groben aufzuschreiben, was genau passieren soll, wenn man dies oder jenes drückt / einstellt. Eben die if / else Verzweigungen sollten gründlich überdacht werden.

HttpListenerContext decode/encode Umlaute

Liest man die Url aus dem HttpListenerContext, die Umlaute wie äöü enthält, so sieht das ungefähr so aus:

aus süß wird s%FC%df.

Abhilfe schaft da die Klasse HttpUtility:
[crayon-677df06fa7816202256018/]
möchte man zurück encodieren macht man einfach :
[crayon-677df06fa781b080027632/]

Referenztypen, Wertetypen

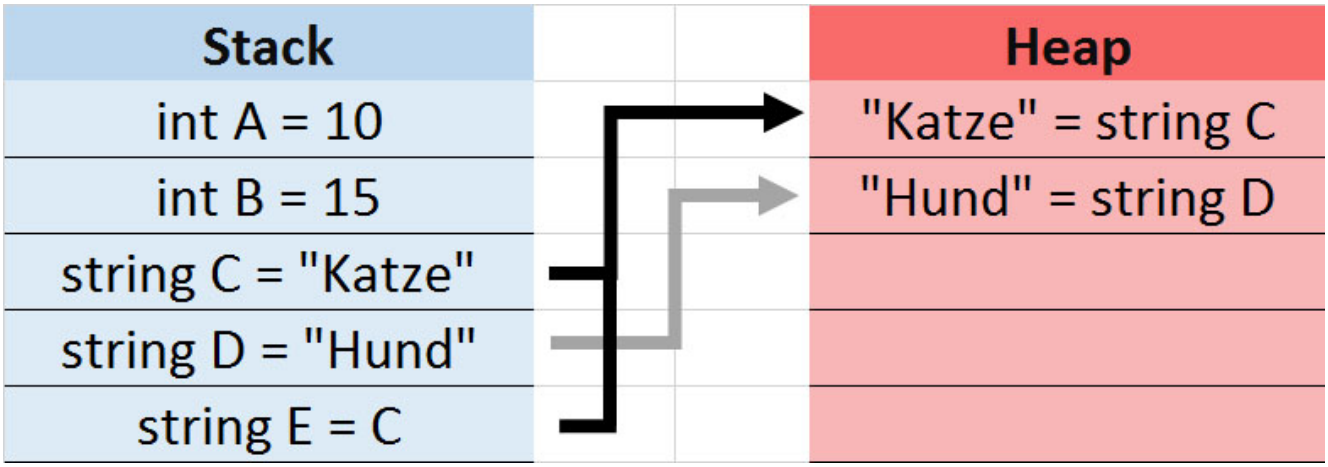
Stack und Heap sind Teil des Arbeitsspeichers, die zu mindestens unter C# vom Kompiler automatisch verwaltet wird.

Beim **Stack** (dt. Stapel) werden die Daten quasi aufeinander gelegt. Wenn Speicher freigegeben werden kann, so wird dies auch von oben heraus getan (LIFO Prinzip). Durch dieses Prinzip ist der Stack sehr schnell in seiner Arbeitsweise. Verlässt der Programmcode die geschwungene Klammer { }, werden sämtliche innerhalb angelegte Variablen aus dem Stack entfernt. Obwohl der Speicher sehr gering ist, wächst und schrumpft er während des Programmablaufs.

Der **Heap** ist nicht so strukturiert wie der Stack, dort liegen die Daten quasi durcheinander und werden vom Stack aus per

Zeiger gefunden. Deswegen ist der Heap auch deutlich langsamer in seiner Arbeitsweise.

Was in den Stack und was in den Heap gelangt, bestimmen unter anderem die Datentypen. Weiter beinhaltet der Stack auch die Zeiger der Variablen auf den Heap.



Stack – Wertetypen:

• Alle numerischen Datentypen
• Boolean, Char und Date
• Alle Strukturen, auch wenn ihre Member Verweistypen sind
• Enumerationen, da der zugrunde liegende Typ immer SByte, Short, Integer, Long, Byte, UShort, UInteger oder ULong ist

Heap – Referenztypen, Verweistypen:

• String
• Alle Arrays, auch wenn ihre Elemente Wertetypen sind
• Klassentypen, z.B. Form
• Delegaten
• reine Objekte

Das Verschieben der Daten vom Stack (Wertetyp) zu Heap (Referenztyp) bezeichnet man als **boxing**
[crayon-677df06fa796b594207968/]
und vom Heap zu Stack als **unboxing**

[crayon-677df06fa796f480981611/]

Auch beim **casten** können die Daten vom Heap zum Stack geschoben werden:
[crayon-677df06fa7970916485970/]

Quelle:

Understanding Boxing and Unboxing