FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Einführung in C#
Übungen

Prof. Dr. rer. nat. Alexander Voß

FH Aachen
University of applied Sciences

16.06.2020

Allgemeine Informationen

Achtung

In jedem Kapitel gibt es repräsentative Übungsaufgaben und auch Beispiellösungen.

Machen Sie nicht den Fehler und gucken sich nur die Lösungen an, nur um dann festzustellen, dass das ja einfach erscheint.

Es ist nicht das unbedingte Ziel, alle Aufgaben zu bearbeiten, sondern sich lieber mit den jeweils unbekannten oder unsicheren Gebieten zu beschäftigen.

Schließen Sie die Aufgaben zu Hause ab und bringen Sie offene Fragen zur nächsten Veranstaltung bzw. zum Chat mit.

0x01 – Übungen

’Aspal’

Erste Schritte in der IDE:

- Laden Sie die Code-Snippets von der Ilias-Seite und entpacken Sie die zip-Datei in einem beliebigen Verzeichnis.
- Starten Sie Visual Studio und/oder Ihre IDE, öffnen Sie die Solutions-Datei und selektieren Sie das Projekt A_HelloWorld als Startprojekt.
- Starten Sie das Programm.

0x01 – Übungen

’Agria’

- Starten Sie Ihre IDE, laden Sie die Snippet-Solution und selektieren Sie A_HelloWorld_Explained als aktives Projekt.
- Übersetzen Sie das Projekt einmal im Release- und einmal im Debug-Modus und führen Sie es einmal ohne und einmal mit Debugger aus.
- Setzen Sie einen geeigneten Debug-Punkt und führen Sie das Projekt Schritt für Schritt aus.
- Räumen Sie das Projekt auf (Clear) und Übersetzen Sie es komplett neu (Rebuild).
- Klammern Sie einen Codebereich in “#region Beschreibung” und “#endregion".
- Schliessen Sie einzelne Fenster, öffnen Sie diese wieder neu und arrangieren Sie sie auf dem Bildschirm nach Wunsch.

0x01 – Übungen

’Aspein’

- Legen Sie ein eigenes Projekt bzw. eine neue Solution mit Projekt in Ihrer IDE an. Dazu gibt es zwei übliche Möglichkeiten der Organisation:
  - Sie legen neue Projekte, z.B. Übungsaufgaben, als Teil der Snippet-Solutions an. Dann können Sie gut zwischen den Snippets und Ihren Projekten hin- und her wechseln.
  - Sie legen neue Projekte in einer eigenen Solution an und starten Ihre IDE ein zweites Mal.

0x01 – Übungen

’Ablium’

- Erzeugen Sie einen Syntaxfehler im Code, z.B. durch Löschen eines Semikolons.
- Beobachten Sie, wie und wo der Fehler in der IDE angezeigt wird.
- Korrigieren Sie den Fehler wieder.
- Schreiben Sie “Console.Wr” (mit Punkt nach Console) in eine neue Zeile und wählen Sie WriteLine im auftauchenden Fenster aller möglichen Methoden aus.
- Benennen Sie eine Variable oder Funktion um, nutzen Sie die Refactoring-Funktionalität Ihrer IDE.

0x01 – Übungen

’Agroisil’

- Generieren Sie eine Liste von Zufallszahlen >=1 und <= 10 und geben Sie diese auf die Console aus.
- Erzeugen Sie ein Array und speichern Sie alle generierten Zahlen zunächst darin und geben Sie danach alle Zahlen aus.
- Nutzen Sie String-Interpolation.
- Erzeugen Sie ein zweites Array, dass ausschließlich die geraden Zahlen des ersten Arrays enthält.

0x01 – Übungen

’Awhya’

- Legen Sie ein neues Projekt an und berechnen Sie die angegebene Summe auf verschiedene Weise. Verwenden Sie zunächst den Datentyp double.

  Summe = 1/1 – 1/2 + 1/3 - 1/4 ± … ± 1/9999 – 1/10000

- Berechnen Sie die Summe
  - von links nach rechts
  - von rechts nach links
  - durch die Addition der Teilsummen der positiven und negativen Terme.
- Was ist der Grund für die unterschiedlichen Ergebnisse?
- Verwenden Sie nun den Datentyp decimal.

0x01 – Übungen

’Astren’

- Legen Sie ein neues Projekt zur Berechnung der Potenz an.
- Berechnen Sie b^n (b hoch n) für feste natürliche Zahlen b und n.
- Schreiben Sie eine Funktion Pot, die b und n übergeben bekommt und das Ergebnis zurückgibt. Geben Sie es dann aus.

0x01 – Übungen

’Ablaria’

- Legen Sie ein neues Consolen-Projekt an zur Berechnung von Primzahlen.
- Berechnen Sie alle Primzahlen bis zu einer festen natürlichen Zahl n und geben Sie diese aus.

0x01 – Übungen

’ Athua’

- Legen Sie ein neues Consolen-Projekt an zur Berechnung der Fibonacci-Folge (f_n).
- Berechnen Sie die ersten Fibonacci-Zahlen bis zu einem festen Index n.
  $f_0 = 0, f_1 = 1, f_n = f_{n-2}+f_{n-1}$ für $n>1$
  nicht-rekursiv in einer Schleife.
- Formulieren Sie die Schleife als for, do-while und while-Schleife.

0x02 – Übungen

’Breau’

Verstehen Sie den Unterschied zwischen ref, out und params.

- Legen Sie ein neues Consolen-Projekt an.
- Schreiben Sie eine Swap-Funktion mit zwei initialisierten int-Argumenten, die diese Parameter tauscht.
- Schreiben Sie eine Funktion, um eine übergebene Variable mit einem Array mit zufälligen Inhalten zu füllen. Wählen sie eine beliebige Länge.
- Schreiben Sie ein Funktion, die mit beliebig vielen int-Werten aufgerufen werden kann, das Maximum ermittelt und dieses zurückgibt.

0x02 – Übungen

’Basinham’

Mustererkennung in einem String.

- Legen Sie ein neues Consolen-Projekt an.
- Schreiben Sie eine Funktion FindAll, die alle vorkommenden Muster in einem String findet und diese Menge als Feld von Positionen zurückgibt.

Entscheiden Sie selbst, ob Sie überlappende Muster finden oder nicht.

0x02 – Übungen

’Bleersora’

Schreiben Sie ein Programm, dass einen Text daraufhin überprüft, ob es sich um ein Palindrom handelt ("Na, Freibierfan!").

- Legen Sie ein neues Consolen-Projekt an.
- Lesen Sie zunächst den Text von der Konsole ein und überprüfen Sie dann in einer eigenen Methode.
- Ignorieren Sie Leerzeichen und Satzzeichen (s.o.).
- Beispiele für Palindrome findet man z.B. unter http://www.gnudung.de/kram/sprache/palindrom.htm

0x02 – Übungen

’Babuin’

Schreiben Sie eine Klasse Kontakt, die einen Namen und ein Alter enthält.

- Legen Sie ein neues Consolen-Projekt an.
- Implementieren Sie einen Default-Konstructor.
- Implementieren Sie einen Konstructor, der einen Namen und ein Alter zur Initialisierung übergeben bekommt.
- Implementieren Sie Properties für den Namen und das Alter.
- Implementieren Sie eine ToString-Methode zur Ausgabe.

0x02 – Übungen

’Bloitol’

Schreiben Sie eine Klasse Buch.

- Legen Sie ein neues Consolen-Projekt an.
- Implementieren Sie geeignete Konstruktoren.
- Implementieren Sie Properties Isbn, Titel und Autor.
- Implementieren Sie eine ToString-Methode zur Ausgabe.

0x02 – Übungen

’Boudadillo’

Objekte aus Dateien generieren und die Ausgabe formatieren.

- Legen Sie ein neues Consolen-Projekt an und lesen Sie die Datei “Bundesliga-0.txt” aus den Snippets ein.
- Schreiben jede Zeile in eine zweite Datei, allerdings in umgekehrter Reihenfolge (letzte Zeile zuerst, nicht die Zeilen selber rumdrehen).
- Entwerfen Sie eine Klasse “Verein” mit geeigneten Eigenschaften, Methoden und Properties, um den Verein und eine Tabelle zu modellieren.
- Ermitteln Sie für jeden Verein die Punkte und Tore am Ende der Saison, am besten in einer eigenen "Tabelle"-Klasse.
- Geben Sie die Vereine und die Tabelle “schön” formatiert erst auf der Konsole, dann in eine Datei aus.

0x02 – Übungen

’Breka’

Entwerfen Sie eine Polynomklasse, die ein Polynom mit reellen Koeffizienten repräsentiert.

- Legen Sie ein neues Consolen-Projekt an, sowie
- einen Konstruktor, der eine Koeffizientenmenge als Parameter bekommt,
- ein Property degree, das den Grad des Polynoms liefert,
- eine Methode eval zum Auswerten an einer Stelle x,
- ein Funktion "Mul" zum Multiplizieren mit einem Skalar, und eine Funktion "Plus" zum Addieren zweier Polynome,
- eine Boolsche Funktionen isZero, die testet, ob es sich um das Null-Polynom handelt.

0x03 – Übungen

’Cluuhmore’

Entwerfen Sie eine Klasse Komplex.

- Legen Sie ein neues Consolen-Projekt an.
- Überlegen Sie sich geeignete Konstruktoren und Properties, damit Sie auf Real- und Imaginärteil zugreifen können.
- Implementieren Sie Operatoren + - * / für Addition, Subtraktion, Multiplikation und Division sowie einen Operator ==.
- Implementieren Sie einen Indicer, der für den Index 0 den Real-Teil und für Index 1 den Imaginär-Teil liest bzw. setzt.
- Überschreiben Sie die Methode ToString.
- Implementieren Sie das Interface IComparable.

0x03 – Übungen

’Chillmond’

Entwerfen Sie eine Klasse Bruch (analog zu Cluuhmore/Komplex).

- Legen Sie ein neues Consolen-Projekt an.
- Überlegen Sie sich geeignete Konstruktoren und Properties, damit Sie auf Nenner und Zähler zugreifen können.
- Implementieren Sie Operatoren + - * / für Addition, Subtraktion, Multiplikation und Division sowie einen Operator ==.
- Implementieren Sie einen Indicer, der für den Index 0 den Zähler und für Index 1 den Nenner liest bzw. setzt.
- Überschreiben Sie die Methode ToString.
- Implementieren Sie das Interface IComparable.

0x03 – Übungen

’Cuetiva’

Schreiben Sie eine Kontakt-Verwaltung als Consolen-Projekt.

- Schreiben Sie eine Klasse Kontakt, die einen Namen (string) und eine Menge von EMail-Adr. enthält.
- Bereiten Sie ein Interface IReport mit einer Funktion MonatsReport vor.
- Leiten Sie von Kontakt eine Klasse Kunde ab, die einen Kontostand enthält, und eine Klasse Lieferant, die eine Bankverbindung (string) enthält.
- Implementieren Sie jeweils für Kunde und Lieferant das Interface mit einer einfachen Ausgabe.
- Erstellen Sie eine Liste für Kontakte und füllen diese mit einigen fiktiven Kunden und Lieferanten auf. Rufen Sie nun die Funktion MonatsReport für Ihre Kontakte auf.
- Tipp: In Kontakt ist MonatsReport noch abstrakt.

0x03 – Übungen

’Caros’

- Legen Sie ein neues Consolen-Projekt an und entwerfen Sie Klassen, um einen (Mini)Zoo zu repräsentieren.
- Entwerfen Sie eine Basisklasse Tier mit wichtigen Eigenschaften, die jedes Tier in einem Zoo hat.
- Leiten Sie Klassen für bestimmte Arten Ihrer Wahl (Vögel etc.) mit speziellen Eigenschaften ab.
- Entwerfen Sie weiterhin eine Klasse Zoo, die alle Tiere in einem Zoo enthält und die welche via += hinzufügen oder via -= löschen kann.
  Tipp: https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/operators/operator-overloading
- Geben Sie alle Tiere aus.

0x03 – Übungen

’Catoria’

Schreiben Sie eine Klasse (FIFO)Queue für ganze Zahlen.

- Legen Sie ein neues Consolen-Projekt an.
- Implementieren sie Funktionen zum Hinzufügen und zum Entfernen von Einträgen, sowie
- eine Ausgabe, die alle Elemente der Queue ausgibt und dabei leert.

0x03 – Übungen

’Castecete’

- Ändern Sie die Klasse Queue aus Aufgabe 'Catoria' zu einer generischen Klasse, so dass Sie beliebige Typen ablegen können.

0x03 – Übungen

’Caféteria’

- Machen Sie auch mal ne Pause…

0x04 – Übungen

’Dinoît’

- Legen Sie ein neues Consolen-Projekt an und definieren Sie ein Delegate, so dass Sie Funktionen der Art

  f(int,int)->int

  aufrufen können.
- Definieren Sie Funktionen zur Addition und Multiplikation zweier Integer-Zahlen, deren Signatur zum Delegate passt. Rufen Sie beide Funktionen über ein Delegate mit Testwerten auf.

0x04 – Übungen

’Dirac’

- Legen Sie ein neues Consolen-Projekt an und definieren Sie ein Feld von Delegates, initialisiert mit den Funktionen sin, cos, tan.
- Geben Sie eine Wertetabelle von 0..2Pi in der Form

  x sin(x) cos(x) tan(x)
- in 0.1 Schritten aus (rufen Sie dazu die Funktionen im Feld auf).

0x04 – Übungen

’Daugnge’

- Legen Sie ein neues Consolen-Projekt an und schreiben Sie eine Funtion eval, die Sie mit einem Lambda-Ausdruck f und einem double-Wert x aufrufen können, und die f(x) zurückgibt.

0x04 – Übungen

’Dallækur’

- Legen Sie ein neues Consolen-Projekt an und schreiben Sie jeweils einen Lambda-Ausdruck, der
  - drei double-Werte addiert und zurückgibt,
  - testet, ob ein int-Wert in einem Intervall [a,b] liegt.

0x04 – Übungen

’Dhernuluhm’

- Legen Sie ein neues Consolen-Projekt an und implementieren Sie ein Newtonverfahren (siehe Wikipedia) zur Nullstellensuche als Funktion, die Sie mit Lambda-Ausdrücken für f und f’ aufrufen können.
  Wählen Sie als Funktion $f(x) = 1 - a/(x^2)$ mit a=2 (siehe auch Snippet zu Delegates) und rechnen Sie so die Wurzel von a (z.B. für 2) aus.

Anmerkung 1: $f(x)=0 <=> a=x^2$

Anmerkung 2: Das Newtonverfahren bekommt zwei Funktionen f und f'. Sie sollen in dem Newtonverfahren die Funktion *nicht* differenzieren (ausser Sie möchten unbedingt). Stattdessen geben Sie die Ableitung mit an.

0x05 – Übungen

'Elunore'

- Legen Sie ein neues Consolen-Projekt an und implementieren Sie eine enumerator-Funktion, so dass folgender Aufruf alle Zeichen des übergebenen Strings rückwärts ausgibt:

```
foreach (char c in MyReverse("Hallo")) {
  Console.Write(" c = "+c);
}
```

- Ausgabe:

```
c=o c=l c=l c=a c=H
```

0x05 – Übungen

’Esari’

- Legen Sie ein neues Consolen-Projekt an und implementieren Sie eine enumerator-Funktion, die nacheinander alle Primzahlen bis zu einer übergebenen Schranke berechnet.

0x05 – Übungen

’Elfshys’

- Legen Sie ein neues Consolen-Projekt an und erweitern Sie die Aufgabe 'Basinham'. Finden Sie alle Vorkommen eines Musters (p) in einem String (s) und nutzen Sie dazu yield bzw. IEnumerable. Das Main-Programm besteht aus den Zeilen:

```
string s = "012mmmm7890mmm", p = "mm";
foreach (int n in findAll(s, p))
    Console.WriteLine(n);
```

- Die Ausgabe dieses Main-Programmes ergibt:

```
3  5  11
```

0x05 – Übungen

’Emberstorm’

- Legen Sie ein neues Consolen-Projekt an. Erzeugen Sie eine Menge von ganzzahligen Zufallszahlen, z.B. 20 Stück, und
- suchen Sie mittels LINQ alle ungeraden Zahlen, bzw.
- ermitteln Sie mittels LINQ die Gruppen der Zahlen mit gleichem Rest bei der Division durch 5 und geben sie die Größe der jeweiligen Gruppe aus.

0x05 – Übungen

’Ebonbrook’

- Legen Sie ein neues Consolen-Projekt an und lesen Sie die Beispieldaten aus simple.xml, https://msdn.microsoft.com/en-us/library/ms767602(v=vs.85).aspx ein. Da wir XML noch nicht behandelt haben, betrachten Sie die Datei als Textdatei mit Tags und Inhalten. Überlegen Sie sich eine geeignete Datenstruktur.
- Suchen Sie nun mittels LINQ in ihrer Struktur alle “Waffles”.

0x06 – Übungen

’Felanathemar’

- Legen Sie ein neues WindowsForm-Projekt an.
- Ziehen Sie nacheinander ein Label, einen Button, eine Listbox und eine TextBox aus dem Tools-Fenster auf die Form.
- Benennen Sie die Controls um und geben Ihnen einen anderen Namen, verändern Sie Ihre Eigenschaften und verstehen Sie, wie diese Eigenschaften funktionieren. Betrachten Sie insbesondere: BackColor, ForeColor, Font, (Name), Anchor, Dock, Location, Size, Enabled, Visible, Text (nicht bei einer ListBox) und TextAlign (ebenfalls nicht ListBox).
- Betrachten Sie die wesentlichen EventHandler OnChange, OnClick.

0x06 – Übungen

’Fyilion’

- Legen Sie ein neues WindowsForm-Projekt an und füllen Sie Zufallszahlen in eine Listbox.
- Fügen Sie einen Event-Handler hinzu, so dass bei Auswahl eines Feldes in der ListBox (Selektion) der entsprechende Eintrag in einer eigenen TextBox angezeigt wird. Hinweis: Das Object an der Stelle des aktuellen Index in den Elementen der ListBox muss ggf. „gecastet“ werden, etwa so:
(string)(this.listBoxMain.Items[index])

0x07 – Übungen

’Gughdarim’

Starten Sie ein eigenes kleines GUI-Projekt (WinForms oder WPF, folgt nächste Woche). Wir werden auch Zeit haben, um daran weiterzuarbeiten. Beispiele (eigene Ideen sind auch in Ordnung, unterschätzen Sie den Aufwand aber nicht):

- 17 und 4 (Karten siehe z.B. Snippets),
- Taschenrechner (einfache Version, oder mit UPN),
- 4-Gewinnt
- “Game of Life” (Wikipedia - Conways Spiel des Lebens).

Mögliche Ausbaustufen:

- Spielfeld dynamisch aufbauen (GUI-Elemente zur Laufzeit anlegen, siehe Snippets).
- Model-View-Control-Pattern nutzen (Ansatz siehe z.B. Snippets).
- Gegen einen Computergegner spielen.

0x08 – Übungen

’Hazelspell’

'Gughdarim' in WPF

- Überlegen Sie, ob Sie Ihr GUI-Projekt in WPF implementieren möchten. Wenn Sie eine sinnvolle Trennung zwischen Daten, Logik und Ansicht zugrunde gelegt haben, sollten sich die Änderungen vorwiegend auf die Ansicht beschränken (Benchmark Ihrer Architektur!).
- Alternativ legen Sie ein neues GUI-Projekt in WPF an.

0x09 – Übungen

’Isolon’

Projekt

- Arbeiten Sie an Ihrem GUI-Projekt weiter.

0x0a – Übungen

’Jivalune’

Threads, ggf. Mutex, Lock

- Summieren Sie die Zahlen 1..100000 (oder kleiner, je nach Leistungsfähigkeit Ihres Rechners) Thread-sicher in einer parallelen Variante mit Threads.

0x0a – Übungen

'JieThalor'

Threads, ggf. Mutex, Lock

- Schreiben Sie eine parallele, sichere Variante der summierten (bzw. zusammen-gesetzten) Sehnentrapezformel (siehe https://de.wikipedia.org/wiki/Trapezregel) bei vorgegebener Anzahl Teilintervalle.

0x0a – Übungen

’JialAnore’

Threads, Mutex, Lock

- Suchen Sie im Bereich der Zahlen 1…100000 (oder kleiner, je nach Leistungsfähigkeit Ihres Rechners) parallel nach der Zahl mit der größten Menge an Teilern. Bei Gleichstand ist die davon größte Zahl gesucht.
  Überlegen Sie sich dazu eine geeignete Parallelisierungsstrategie.
- Können Sie Ihre Strategie verbessern? Denken Sie an Laufzeit, Anteil parallelem zu seriellem Code etc.

0x0a – Übungen

’Jalmel’

Threads

- Das Projekt ‘Z_Sieve’ enthält eine serielle Variante des Sieb des Eratosthenes zur Ermittlung aller Primzahlen bis zu einer gegebenen Grenze sowie Methoden zur Zeitmessung. Überlegen Sie sich eine parallele Variante und vergleichen Sie die benötigte Zeit.

0x0b – Übungen

’Krulnur’

Master-Worker-Pattern (manchmal Master-Slave oder Map-Reduce)

- Als Aufgabe definieren wir den Test einer Zahl, ob sie eine gewisse Eigenschaft besitzt, etwa prim, gerade oder vollkommen zu sein (wählen Sie selbst).
- Sammeln sie "Aufgaben" (z.B. Zahlen 1..1000 oder 1000 Zufallszahlen) in einem Pool und arbeiten Sie diese Aufgaben in einem Master-Worker-Pattern ab.
- Grob gesagt besteht ein Master-Worker-Pattern aus einer festen Menge von Worker-Threads, die sich jeweils aus diesem gemeinsamen Pool von Aufgaben die nächste nehmen, sie bearbeiten, das Ergebnis in eine Ergebnismenge ablegen und das dann solange wiederholen, bis der Pool leer ist.
- Beachten Sie, dass der Pool und ggf. die Ergebnismenge eine gemeinsam genutzte Resource darstellt und entsprechend geschützt werden muss. Wer mag, kann hier auch Datenstrukturen aus dem Concurrent Namespace nutzen (selbst lesen).

0x0b – Übungen

’Krarogg’

Tasks

Modifizieren Sie eine Ihrer Lösungen aus Übung 0x0a wie folgt:

- Nutzen Sie explizit Tasks anstelle von Threads.