C# Dag 2 - sprogkonstruktioner

De mest almindelige
Sprogkonstruktioner i C#
(meget er som Java)

Emner i dag

- Namespaces
- Value types og reference types
- Standard value types
- Variabler
- Konvertering/cast
- Parameteroverførsel
- static, const, readonly
- Properties
- Betingede sætninger (if, switch)
- Løkker
- Exceptions
- Arrays

C#-program

- En samling klasser, hvoraf én har en Main-metode (i hvert fald i consol programmer).
- Initialisering: enhver variabel skal initialiseres før brug.
- Felter, der ikke eksplicit initialiseres af programmøren, bliver automatisk nulstillet.

Simpelt C# konsolprogram

```
using System;                              // Bemærk: using svarer til
using System.Collections.Generic;          // import i Java
using System.Linq;
using System.Text;

namespace ConsoleApplicationTest           // namespace svarer til
{                                          // package i Java
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Test");// Svarer til
            Console.ReadLine();            // System.out.println
        }
    }
}
```

Namespace

C# bruger namespace til at organisere koden i moduler. Svarer til C++'s namespace og Pascals unit eller som package systemet i Java.

Eksempel på anvendelse:

```
using System.Drawing;
```

giver adgang til alle erklæringer i namespace System.Drawing.

Erklæring af namespace

```
namespace ErhvervsakademiAarhus {

// alt erklæret her ligger i
// namespace ErhvervsakademiAarhus

...

}
```

Brug af namespace

Font-class kommer fra namespace ”System.Drawing”:

```
using System.Drawing;
…
Font f=new Font(”Arial”,18,FontStyle.Bold);
textBox1.Font=f;
```

Man kan også gøre sådan – uden ”using”, men koden bliver lidt ”tung”

```
System.Drawing.Font f=
    new System.Drawing.Font(”Arial”,18,
                        System.Drawing.FontStyle.Bold);
textBox1.Font=f;
```

Hierarki af namespaces *

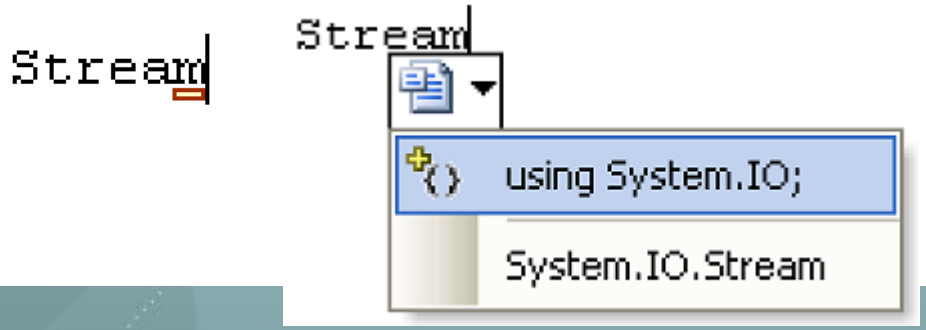
```
namespace Aabc {
// any element declared here belongs to
// namespace Aabc

   namespace Dmu {
    // any element declared here belongs to
   // namespace Aabc.Dmu

   }

}
```

Namespace intellisense

- Visual Studio kan ofte selv foreslå et nødvendigt namespace
- Skriver man f.eks. Stream uden at have

```
using System.IO;
```

vises en lille markering under m, som man kan trykke på og automatisk få indsat using eller eksplicit namespace:



Typer i C# - primitive types?

I C# er der ikke som sådan en primitive type, som I kender fra int I Java.

Alt er en klasse/object

Der er to typer af klasser – value types og reference types.

(i.e. i Java har vi jo både int og Integer – ikke sådan i C#, hvor int faktisk er en klasse)

Value typer og reference typer

- Valuetyper er typer såsom int, char mv. Desuden er Struct en slags record lignende valuetype. **De indeholder en konkret værdi**.
- Reference typer er alle slags klasseinstanser af f.eks. dine egne klasser eller indbyggede klasser, som ikke er valuetyper.
- Se iøvrigt i:

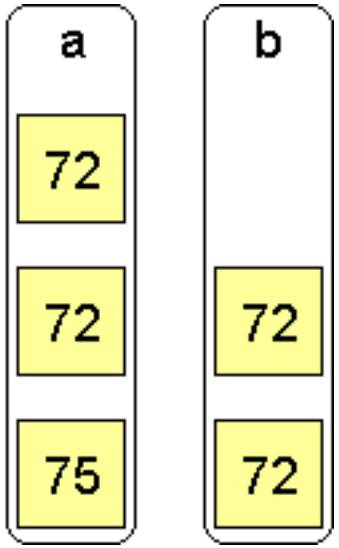
  [C# for Java Developers], Appendix C

  http://media.wiley.com/assets/264/21/0764557599_bonus_AppC.pdf

Value og reference type adfærd

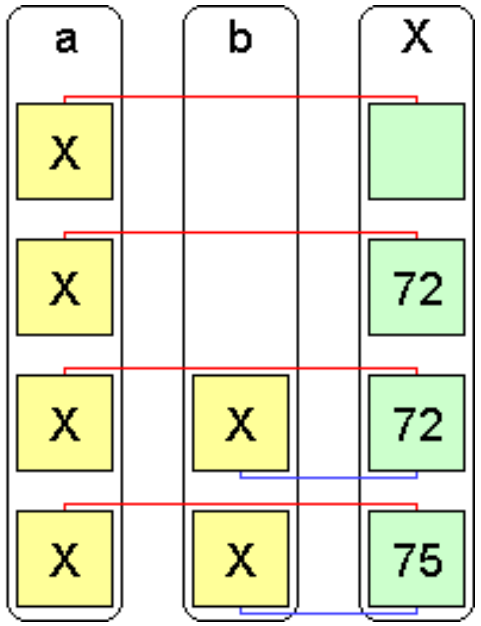
Value type

```
int a = 72;

int b = a;

a = 75;
```



Reference type (X er den nye værdi)

```
Person a = new Person();

a.Height = 72;

Person b = a;

a.Height = 75;
```



Value type brugt som objekt

I C# kan en value type bruges som et objekt, f.eks. en int (fordi ALT er klasser/objekter, også int):

```
int i=25;
...
string s=i.ToString(); //Kald metode
...
int m=77;
object o=m;  //Cast til Object. Boxing
int j=(int)o;
```

Variabler

```
public void DefineVariables() {
    int i;
    int j = 0;

    double[] d1 = null;
    double[] d2 = new double[10];
    double[] d3 = new double[] { 2, 3, 4 };
    double[] d4 = { 3, 6, 9 };
    double[] d5 = new[] { 55.0, 77.0, 11.4 };

    string s1 = "Hello world";
    string s2 = @"c:\user\pictures";
    string s3 = $"Der er {3} spor efter kattepoter i min lagkage";

    MyClass q1 = new MyClass(); // eksplicit type
    var q2 = new MyClass();     // implicit type
}
```

Type inference

- `var q2 = new MyClass();  // implicit type`

Predefinerede value typer

- Integer typer: int (32 bit), long (64 bit) og flere (sbyte, short, byte, ushort, uint, ulong samt med længdeangivelse – f.eks. UInt32)
- Floating point typer: double og Double (samt float)
- Decimal: decimal og Decimal (128 bit high precision type)
- Boolean: bool og Boolean (false, true)
- Tegn: char og Char (16 bit Unicode character)
- System.Drawing.Point

Konverteringer (widening)

Implicitte konverteringer - generelt fra lavere præcision til højere præcision:

byte → int

float → double

Eks:

```
byte b=5;     int i;     …     i=b; // ok
float f=3.7; double d; …     d=f; // ok
```

Samme ved typer, der nedarver

Konverteringer (narrowing)

Eksplicit konvertering - fra højere præcision til lavere præcision skal foretages ved eksplicit angivelse

Eks.

```
int i=5; byte b;  …
// b=i; er ulovlig
b=(byte) i; // ok

double d=3.7; int i; …
// i=d; er ulovlig
i=(int) d; // ok
```

Samme ved typer, der nedarver

Enumerations (enum)

- En brugerdefinet integer type
- Alle værdier nævnes i definition af enumeration
- Eksempel:

```
enum Direction
{
        Up = 0,
        Down = 1,
        Left = 2,
        Right = 3
}
```

Predefinerede reference typer

- object = System.Object (den ultimative superklasse)
- string = System.String (Unicode character string)

"\" er escape-tegn i strenge. Kan undertrykkes med @ foran strengkonstanten:

```
string path1="c:\\myfiles\\brev.doc";
string path2=@"c:\myfiles\brev.doc";

//Så her er de to paths det same.
```

Parameteroverførsel

Følgende typer parameteroverførsel:

- by value (default – som i Java)
- by reference
- output variabel (reference)
- input variabel (reference)
- variabelt antal parametre

Value parameter

- Værdi kopieres fra aktuel til formel parameter
- Hvad udskrives til konsol?

```
void F(string a) {
    a = ”changed”;
}
…
string a1 = "initial”;
F(a1);
Console.WriteLine(a1);
```

Reference parameter

- Reference til værdi kopieres fra aktuel til formel parameter
- Hvad udskrives til konsol?

```
void F(ref string a) {
    a = ”changed”;
}
…
string a1 = "initial”;
F(ref a1);
Console.WriteLine(a1);
```

NB: Generelt skal man nok passe på med at lade metoder ændre input parametrene!!

Reference parameter

- Eksempel på tavlen
- Eksempel søgning efter person
- eller Int32.TryParse
- in og out-parametre

PAUSE

Objekter og typer

- En class erklæres som i Java
- En fil kan indeholde flere klasser, derfor ingen forbindelse mellem class navn og filnavn **(men generelt vil jeg anbefale en klasse per fil og så give dem det samme navn som filnavnet, altså Person ligger i Person.cs filen)**
- Tilsvarende ingen forbindelse mellem namespace og filnavn, som kan være forskellige.
- (så flere filer kan have samme namespace – som også er almindeligt. Tænk på det som en package)

Static

- static felter og metoder - som Java
- static constructor kan initialisere en static var - som Java
- en property kan være static
- Static ≈ hører til klasse, ikke instanser. Refereres via klassenavn, ikke instans

NB: Ikke alle anbefaler brugen af static, måske med undtagelse af konstanter og til små test metoder som skal kaldes fra main. Nyere programmeringssprog som f.eks. Kotlin har faktisk ikke engang et static keyword

Konstanter

Erklæres med nøgleordet const

```
public class Car {
    private const int maxSpeed = 100;
    ..
}
```

- En const er implicit static – eksisterer altid
- En const *skal* initialiseres i erklæringen
- En const evalueres på compile time

ReadOnly felter (fields)

Erklæres med nøgleordet readonly:

```
public class Car {
    private readonly int maxSpeed = 100;
    ..
}
```

- Readonly felter kan være static eller non-static
- Readonly felter kan initialiseres
  - i erklæringen
  - i en constructor
- Readonly fields evalueres på runtime
- Svarer til final i Java

Property

- property er en C# facilitet som også er kendt fra Visual Basic og Delphi
- Bruges i stedet for get- og set-metoder

Property eksempel

```
class Person
{
   private string cpr; //bemærk private field

   public string CPR  // bemærk public Property
   {
        get { return cpr; }
        set { cpr=value; }  // der kan evt. foretages check
                            // eller kastes en exception
   }
}

Person p=new Person();
p.CPR=”2012131456”;
...
string s=p.CPR;
```

Avanceret property – bemærk setteren

```
class Data {
  FileStream s;

  public string FileName
  {
    set {s=new FileStream(value,FileMode.Create);}
    get {return s.Name;}
  }
  ...
}

Data d = new Data();
d.FileName="myFile.txt"; // skaber ny FileStream i d
```

Automatisk property – ofte anvendt (autoimplemented property)

```
class Person
{
   public string CPR { get; set; }  // easy typing
}

Person p=new Person();
p.CPR=”2012131456”;
...
string s=p.CPR;
```

Expression-bodied members

- Somme tider ses properties skrevet som dette:
- `public int Alder { get => alder; set => alder = value; }`
- Dette er en syntaks kaldet Expression-bodied member
- **Vent** med at bruge den til senere, det forvirrer bare
- Mere her:
- https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/expression-bodied-members

Comments

- En linie:

```
// Dette er en kommentar
```

- Flere linier:

```
/*

. . .

*/
```

- Enkelt linie XML i koden:

```
///XML-kommentar
```

Kontrol sætninger

- Selection sætninger
  - if – else som Java
  - switch
- Repetition sætninger
  - while som Java
  - do – while som Java
  - for som Java
  - foreach

Control structures: if/for/while/foreach

```
public void Controltructures() {
    for (int i = 0; i < 10; ++i)
        Console.WriteLine($"i = {i}");   // bemærk $ for format

    int[] numbers = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    int j = 0;
    while (true) {
        Console.WriteLine($"j = {numbers[j]}");
        if (j++ >= 9)
            break;  //vigtigt at huske!
    }

    foreach(var n in numbers) //bemærk type inference her
        Console.WriteLine($"n = {n}");
}
```

switch sætningen

Eksempel:

```
    //country er en string - er der forskel på
Danmark/Norge her?
    switch (country) {
          case "Danmark":
          case "Norge:
             DanishCall();
             break;  //break er tvunget i C#
          case "Frankrig":
             FrenchCall();
             break;
          default:
             EnglishCall();
             break;
      }
```

foreach sætningen

Eksempel med collection af strings:

```
//tæl antal strenge, som begynder med 'a'
int antal = 0;
foreach (string s in listBox1.Items) {
    if (s[0] == 'a') {
        antal++;
    }
}
```

Eksempel med array:

```
//summer værdierne i arrayet intArray
int sum = 0;
foreach (int value in intArray) {
    sum += value;
}
```

Control statements

- break: hop ud
- continue: fortsæt med næste iteration
- return: ud af metode
- USE SPARINGLY. Efter min mening skal de slet ikke bruges. Men der er jeg nok i mindretal.

Exceptions

Eksempel:

```
try
{
        fileIn = File.OpenRead(oldPath);
        fileOut = File.Create(newPath);
        . . .
}
catch (Exception err)
{
        Console.WriteLine(err.Message);
}
finally
{
        if (fileIn != null) fileIn.Close();
        if (fileOut != null) fileOut.Close();
}
```

Throwing an Exception

- Ved fejl – med en meddelse:

```
throw new ArgumentException("Antal <=0");
```

Eller blot, uden meddelse

```
throw new ArgumentException();
```

Custom Exception class

```
public class DeletedPersonException: Exception
{
  public DeletedPersonException()
  { }

  public DeletedPersonException(string message) :
  base(message)  // bemærk base keywordet
  { }

  public DeletedPersonException(string message,
  Exception inner) : base(message, inner)
  { }
}
```

Exceptions

- INGEN guarded exceptions, dvs. ingen throws erklæring

Arrays

- En-dimensionelt array:

```
int[] temp = new int[10];
temp[1] = 7;
```

- Fler-dimensionelt rektangulært array:

```
int[,] matrix = new int[2,5];
matrix[1,2] = 7;
```

Arrays

- Fler-dimensionelt ”jagged” array: (dvs. hver række kan have forskellige størrelse – se nedenfor)

```
int[][] jagged = new int[3][];
jagged[0] = new int[2];
jagged[1] = new int[5];
jagged[2] = new int[3];
jagged[1][2] = 7;
```

Arrays

- Arrays er ”zero-based”
- Et jagged array er et array af arrays
- Et array nedarver fra klassen System.Array
- Et array har fast længde i modsætning til Collection classes som List.
- CLR udfører bounds checking på arrays

Arrays

- Totalt antal elementer i et array:

```
temp.Length
multi.Length
```

- Længden af dimension `j` i et multi-dimensionalt array

```
matrix.GetLength(j)
```

- Alternativ ved et jagged array:

```
jagged[j].Length
```

Arrays

- Initialisering:

```
char[,] letters = new char
    {{'a','b','c},{'x','y','z'}};
```

  giver et 2 x 3 rektangulært array
- Ofte kan new overspringes:

```
new char
```

  kan udelades i eksemplet herover

Opgaver