GUI1 - Noter

Alexander Rasmussen

28. januar 2018

Indhold

1 Lektion 1 - Intro og .Net arkitektur

1.1 Forberedelse

Forberedelse til første lektion i GUI.

1.1.1 Chapter 1 Introducing C# and .NET - - C# Succintly

C#: er et general-purpose, OO baseret programmeringssprog. Man kan lave ASP.NET, desktop apps og meget andet med C#
.NET: er en platform som har forskellige typer sprog, bl.a. C#. .NET gør brug af CLR som compiler alt til et intermediate Langeuage. Samtidig er det også Framework Class Libraries (FCL). FCL har en del klasser til rådighed som man kan gøre brug af, når der skal laves software.

Der findes meget mere end dette til det, men det er ikke det bogen handler om, det var kun til intro.

```
using System;
class Greetings
{
        static void Main()
        {
        Console.WriteLine("Greetings!");
        }
}
```

Listing 1: Hello World i C#

Static betyder der kun kan være en instans af vores greetings class. Main skal være static, der skal kun være en instans af den. Andre metoder kan omgå den statiske metode, som laver instans members. Det betyder du kan have mange kopier af klasser eller instanser med deres egen metoder. Når man arbejder med C# er det vigtigt at gøre rigtigt brug af namespace for at få mest ud af det. Se eksempel nedenfor på 2

```
using static System.Math;
namespace Syncfusion
{
        public class Calc
        {
                public static double Pythagorean(double a, double
                   b)
                {
                        double cSquared = Pow(a, 2) + Pow(b, 2);
                        return Sqrt(cSquared);
                }
        }
}
```

Listing 2: Hello World i C#

Når man bruger using Sync; og der så ligger noget inde i namepsace som fx calc. så vil man i stedet for at skulle skrive sync.calc.something kunne nøjes med at bruges calc.something.

Der kan også bruges alias til namespaces. **using Crypto = System.Security.Cryptography** Her har vi et alias for det lange navn.

- using static - Gør så koden kan bruge statiske members ind fra den klasse man bruger, uden at have fulde kvalifikationer.

1.1.2 Chapter 2 Coding Expressions and Statements - C# Succintly

C# er et strongly typed sporg hvilket vil sige, jeg ikke kan konvertere mellem forskellige typer som ikke er kompatible. Der er dog måder dette kan gøres på. Se nedenfor på 3 casting som det kendes fra C++ er også en mulighed for valid konverteringer

```
int total = int.Parse("359");
string message = 7.ToString();
```

Listing 3: Eksempler på konvertering C#

Meget syntax minder om C++ og der er rigtigt mange operatore og lignede som er ens. Kapitlet kan være godt at slå op i for syntax hjælp

1.1.3 Chapter 3 Methods and Properties - C# Succintly

Operatoren ?. betyder at man tjekker om venstre siden er = null Slå op i kapitlet hvis der mangler inspiration til refactoring eller bl.a. omkring get og set. Umiddelbart ikke andet som er hver at notere. Exeption handling minder meget om C++

1.1.4 Chapter 4 Writing Object-Oriented Code - C# Succintly

- Inheritance fungere på samme måde for C# som i C++, ved at man kan bruge funktioner fra hvor man nedarver.
- Hvis Access modifiers ikke bliver beskrevet vil de per default være internal og klasse members være private.
- Structs kan ikke nedarve i C# som vi kender det fra C++
- Structs copies by value -> Class copies by reference
- Operatorer er altid staitc
- Implicit - betyder det er en sikker data konvertering
- Explicit - betyder det er en udsikker data konvertering, hvilket betyder du kan miste data. (Du skal her bruge cast operator for at det sker)
- Polymorphism - Fungere med virtual som base class function og så bruger man override til at overskrive funktionerne. Override fungere også som virtual for klasser længere nede i systemet. base.something kalder det funktionen gør i base klassen.
- Abstract classes - Er det sammen som det vi kalder pure virtual i C++. Du skal lave overide på en abstract class med abstract functions.
- interface - Både klasser og structs kan nedarve fra flere interfaces. For interfaces bruger man normalt prefix I. Når man har med interface at gøre minder meget om det samme som Abstract class, men man skal ikke bruger override på metoderne. Interfaces kan nedarve fra andre interfaces, hvis det sker skal den nedarvede klasse implementere alle de nedarvede members.

- Object lifetime - Objekter som structs og enums livs tid er til de går out of scope. En class er det CLR der creater den og det er først når CLR garbage collector cleaner dem op at de forsvinder. Nogle regler hvordan eksekveringen sker undervejs i en konstruktion.
  - Static constructors execute before instance constructors.
  - Static constructors execute one time for the life of the program.
  - Base class constructors execute before derived class constructors.
  - The this operator causes an overloaded constructor that matches the this parameter list to execute first.
  - The base class default constructor executes, unless the derived class uses base to explicitly select a different base class constructor overload.
  - This is not shown in the output of the previous example, but static fields initialize before the static constructor and instance fields initialize before instance constructors.
  - Auto-implemented property initializers, such as Created, initialize at the same time as fields.
  - Properties in object initialization syntax execute last as object initialization syntax is equivalent to populating the property through the instance variable after instantiation.
- Garbage collection er uforudsigeligt, det bliver først kørt når der er brug for det. Man kan bruge Dispose pattern, se mere på side 59-60 omkring det.

Listing 4 viser eksempler på Objekter orienteret C#

```
//Inheritance
public class Calculator { }
public class ScientificCalculator : Calculator { }
public class ProgrammerCalculator : Calculator { }
//Implicit og Explicit
public static implicit operator Complex(double dbl)
{
        Complex cmplx = new Complex();
        cmplx.Real = dbl; return cmplx;
}
// This is not a safe operation.
public static explicit operator double(Complex cmplx)
{
        return cmplx.Real;
}
//Polymorphism
public class Calculator
{
        public virtual double Add(double num1, double num2)
        {
                Console.WriteLine("Calculator Add called.");
                return num1 + num2;
        }
}

public class ProgrammerCalculator: Calculator
```

```
{
        public override double Add(double num1, double num2)
        {
                Console.WriteLine("ProgrammerCalculator Add
                    called.");
                return MyMathLib.Add(num1, num2);
        }
}

public class MyMathLib {
        public static double Add(double num1, double num2)
        {
                return num1 + num2;
        }
}

public class ScientificCalculator: Calculator
{
    public override double Add(double num1, double num2)
    {
        Console.WriteLine("ScientificCalculator Add called.");
        return base.Add(num1, num2);
    }
}

//Abstract classes
public abstract class Calculator
{
        public abstract double Add(double num1, double num2);
}

//Interface
public interface ICalculator
{
        double Add(double num1, double num2);
}
```

Listing 4: Eksempler på konvertering C#

Hvornår vil man bruge et struct fornuftigt. Se eksempel i Listing 5. Grunden til det er en god struct er fordi du kan ske at have en masse matematiske operationer og så er det mere effektivt at have kopier på stacken i stedet for at allokere på heapen.

```
public struct Complex {
 public Complex(double real, double imaginary)
 {
  Real = real;
  Imaginary = imaginary;
 }

 public double Real
 {
  get;
  set;
 }

 public double Imaginary
```

```
{
 get;
 set;
}

public static Complex operator + (Complex complex1, Complex
    complex2)
{
 Complex complexSum = new Complex();
 complexSum.Real = complex1.Real + complex2.Real;
 complexSum.Imaginary = complex1.Imaginary + complex2.Imaginary;
 return complexSum;
}

public static implicit operator Complex(double dbl)
{
  Complex cmplx = new Complex();
  cmplx.Real = dbl;
  return cmplx;
 } // This is not a safe operation. public static explicit
     operator double(Complex cmplx) { return cmplx.Real; } }
```

Listing 5: Eksempler på konvertering C#

1.2 Lektion

1.2.1 Spørgsmål

- Overloading af equals og det med at sætte det = NULL
- Kan man have funktioner i den nedarvede klasse som ikke er i interfaces, men som stadig er public?