

Introduktion til C# og .NET

Xamarin Mobil App Udvikling Modul 3

Modul Indhold

1. Overblik
2. Program flow
3. Kodestruktur
4. Objekt Orienteret Programmering
5. .NET BCL

Overblik I

- Et C# program er opbygget af:
 - Import deklarationer
 - Namespace deklarationer
 - Klasser
 - Metoder
 - Statements og expressions
 - Kommentarer
 - En Main metode

Overblik II

Hello World program i C#:

```
using System;

namespace Hello.World
{
    public class HelloWorld
    {
        public static void Main()
        {
            Console.WriteLine("Hello World");
            Console.ReadKey(); // wait for user to press a key before exit
        }
    }
}
```

Opgave

- Skriv et konsol program, der skriver “Hello Xamarin” ud og venter på at brugeren trykker en tast, før det lukker.
- Tip: F5 kører det projekt i Visual Studio, som er sat til at være *Startup Project*.

Program flow

Program flow - Local declarations

- Lokale deklARATIONER angiver værdier som kun kan/skal bruges i den aktuelle metode.
- Lokal **const** deklaration:
 - Angiver en lokal konstant.
 - Værdi der ikke kan ændre sig.
- Lokal variabel deklaration:
 - Angiver en værdi, der kan ændre sig.

```
// Constants must always have a value
const int theAnswer = 42;
const string yes = "Yes";

// Error: only numbers, bools, strings and
// null references allowed to be const
//const Guid g = Guid.Empty;

// Variables need not be initialized at
// declaration
int index;

// Can declare multiple variables of same type
int hours = 8, minutes = 14;

// References can be set to null (empty
// reference)
string answer = null;

// Invalid: cannot use variable index
// before it has been assigned a value
//int current = index;

// Can assign value after declaration
index = 1;

// Now we can use it
int current = index;
```

Program flow - Expression statements

- En *expression statement* evaluerer en expression og smider resultatet væk.
- Kan kun bruge expressions af typer, der har sideeffekter.
- En *assignment* tildeler en værdi til en variabel.
- Et metodekald eksekverer koden i en metode.
- **new** konstruerer en ny instans af en klasse.

```
// Assignment
int x = Add(1, 1);

// Increment
x++;

// Decrement
x--;

// Method call
Add(2, 3);

// Instantiation
new Example();

// Error: only assignment, call, increment,
// decrement and new object expressions can
// be used as statements
//2 + 2;
```


Program flow - Operatorer I

- Aritmetiske:

+, -, *, /	Sædvanlig plus, minus, gange og dividere.
%	Rest efter heltalsdivision (ikke det samme som modulus).
+	For strenge, bruges til at sætte to strenge sammen.

- Relationelle:

==	Lig med - tjekker om to værdier er ens.
!=	Forskellig fra - tjekker om to værdier er forskellige.
>, >=	Større end og større end eller lig med.
<, <=	Mindre end og mindre end eller lig med.

Program flow - Operatorer II

- Logiske:

<code>x && y</code>	Logisk AND (<code>x == true, y == true</code> → true ellers false). Short circuit eval.
<code>x y</code>	Logisk OR (<code>x == false, y == false</code> → false ellers true). Short circuit eval.
<code>!x</code>	Logisk NOT (<code>x == false</code> → true ellers false).

- Assignment:

<code>x = y</code>	Tildeling - sæt x til samme værdi som y.
<code>+=, -=, *=, /=, %=</code>	Kombineret aritmetik og tildeling.
<code>++x, --x</code>	Prefix increment/decrement - Læg +/-1 til x og returnér resultatet.
<code>x++, x--</code>	Postfix increment/decrement - Læg +/-1 til x og returnér gammel værdi af x.

Program flow - Operatorer III

- Diverse:

<code>x.y</code>	Member access - tilgår medlem i en type eller namespace.
<code>x[y]</code>	Indeks operator - tilgår data i arrays og collections.
<code>typeof(x)</code>	Returnerer typen af x (System.Type instans).
<code>default(T)</code>	Returnerer default værdien for typen T.
<code>c ? x : y</code>	Conditional/ternary operator (<code>c == true</code> → x ellers y).
<code>x is T</code>	Type check - true hvis x er af typen T (eller en underklasse til T).
<code>(T)x</code>	Type cast - konverterer x til typen T - kaster exception hvis ej muligt.
<code>x as T</code>	Type cast - konverterer x til typen T - returnerer null hvis ej muligt.
<code>x ?? y</code>	Null-coalescing operator - (<code>x == null</code> → y ellers x).
<code>x?.y, x?[y]</code>	C# 6.0: member access / indeksering men med null check. Short circuits.
<code>nameof(expr)</code>	C# 6.0: Giver navnet på en variabel, type eller members som en streng.

Program flow - If

Basis syntaks der bruges til at vælge program vej med.

```
int condition = 4;

if (condition == 4 )
{
    Console.WriteLine("The variable was 4");
}
else if (condition < 4)
{
    Console.WriteLine("The variable was less than 4");
}
else
{
    Console.WriteLine("The variable was more than 5");
}

var result = condition == 4 ? "equals 4" : "not 4";
```

Shortcut for If/else er ? notation - også kendt som "ternary operator".

Program flow - Løkker

For loops

While loop

Do while

Foreach

```
var strArray = new string[] { "abc", "def", "ghi" };
for(int i = 0; i < 3; i++)
{
    System.Console.WriteLine(strArray[i]);
}

int count = 0;
while (count < 3)
{
    System.Console.WriteLine(strArray[count]);
    count++;
}

System.Console.WriteLine("waka");
int otherCount = 0;
do
{
    System.Console.WriteLine(strArray[otherCount]);
    otherCount++;
}
while (otherCount == 0);

foreach (String str in strArray)
{
    System.Console.WriteLine(str);
}
```

Program flow - Switch

Godt til enums og tal men kan også benyttes til strings.

Styrken er specielt ikke at glemme cases for tal og enums.

```
static void Main()
{
    Console.WriteLine("Coffee sizes: 1=small 2=medium 3=large");
    Console.Write("Please enter your selection: ");
    string str = Console.ReadLine();
    int cost = 0;

    // Notice the goto statements in cases 2 and 3. The base cost of 25
    // cents is added to the additional cost for the medium and large sizes.
    switch (str)
    {
        case "1":
        case "small":
            cost += 25;
            break;
        case "2":
        case "medium":
            cost += 25;
            goto case "1";
        case "3":
        case "large":
            cost += 50;
            goto case "1";
        default:
            Console.WriteLine("Invalid selection. Please select 1, 2, or 3.");
            break;
    }
    if (cost != 0)
    {
        Console.WriteLine("Please insert {0} cents.", cost);
    }
    Console.WriteLine("Thank you for your business.");
}
```

Program flow - Try/Catch

Bruges når noget går galt.

Du kan catche flere typer Exceptions. Nogle har flere constructors.

Argument Exception er din ven i Xamarin

```
static void Main()
{
    int a = 1;
    int b = -1;

    try
    {
        SomeFunction(a, b);
    }
    catch (ArgumentException aEx)
    {
        System.Diagnostics.Debug.WriteLine(aEx.ParamName);
        System.Diagnostics.Debug.WriteLine(aEx.Message);
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.WriteLine(ex.Message);
    }
}

2
1 reference
static int SomeFunction(int a, int b)
{
    if (a < 0)
    {
        throw new ArgumentException("must be higher than 0", "a");
    }
    if (b < 0)
    {
        throw new ArgumentException("must be higher than 0", "b");
    }
    return a / b;
}
```

Program flow - using statement

- Brugt til automatisk at rydde op efter objekter der implementerer IDisposable.
- Alternativ til try/finally.
- Kalder Dispose() på variabelen når program flow forlader scope.

```
public int ReadFirstLine(string path)
{
    Stream file = null;
    TextReader reader = null;
    try
    {
        file = new FileStream(path, FileMode.Open);
        reader = new StreamReader(file, Encoding.ASCII);
        return file.ReadByte();
    }
    finally
    {
        if (reader != null)
            reader.Dispose();
        if (file != null)
            file.Dispose();
    }
}

public string ReadTextFile(string path)
{
    using (var file = new FileStream(path, FileMode.Open))
    using (var reader = new StreamReader(file, Encoding.ASCII))
    {
        return reader.ReadToEnd();
    }
}
```


Program flow - var keyword

Var er et keyword der bruges hvis noget ønskes implicitly typed.

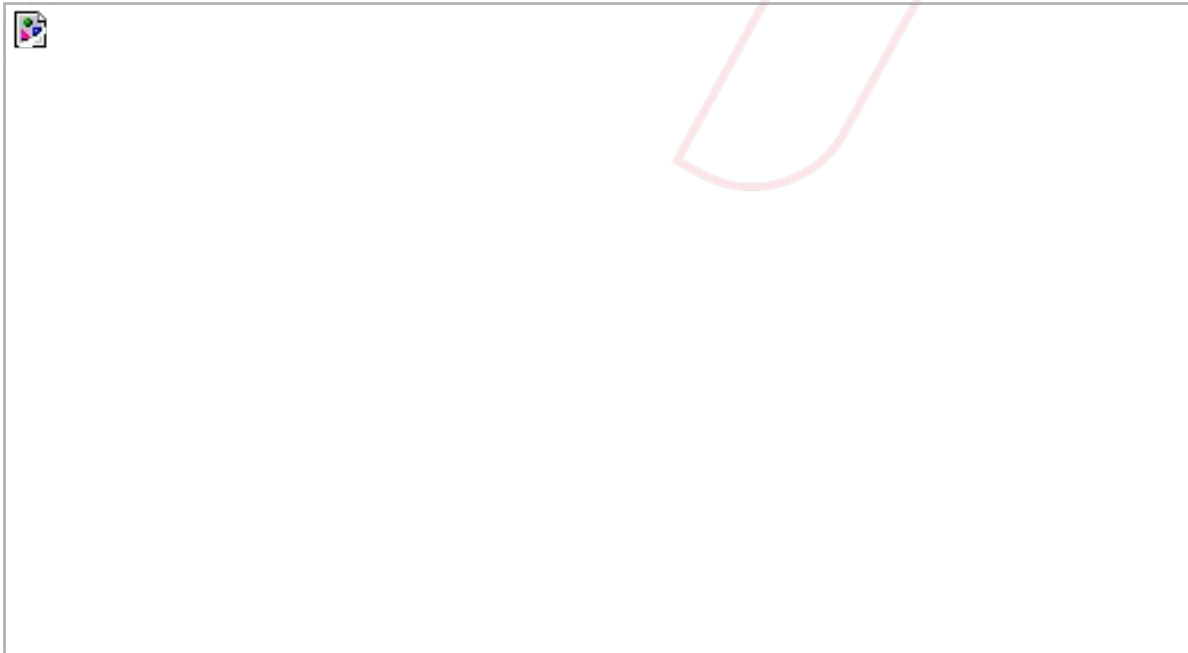
```
var i = 10; // implicitly typed  
int i = 10; //explicitly typed
```

Er funktionelt det samme.

- Kan kun bruges til lokale variable
- Kan ikke initialiseres til null.
- Kan ikke bruges i class scope.
- Kan ikke initialiseres til en expression: `var i = (i = 20);`
- Kun en variabel pr statement
- Brug vars hvor du kan :)

```
// i is compiled as an int  
var i = 5;  
  
// s is compiled as a string  
var s = "Hello";  
  
// a is compiled as int[]  
var a = new[] { 0, 1, 2 };  
  
// expr is compiled as IEnumerable<Customer>  
// or perhaps IQueryable<Customer>  
var expr =  
    from c in customers  
    where c.City == "London"  
    select c;  
  
// anon is compiled as an anonymous type  
var anon = new { Name = "Terry", Age = 34 };  
  
// list is compiled as List<int>  
var list = new List<int>();
```

Kodestruktur



Kodestruktur - using

- Import deklarationer angives med keyword **using**.
- Giver adgang til typer i andre namespaces.
- De fleste indbyggede typer ligger under “System” namespace og underliggende namespaces.
- **C# 6.0:** kan bruge **using static** til at tilgå statiske members i en type uden at foranstille type navn.
- Avanceret: *using kan bruges til at lave alias for en type eller et namespace.*

Kodestruktur - namespaces

- Angives med keyword **namespace**.
- Bruges til at:
 - Undgå kollisioner med andre typer med samme navn.
 - Organisere kode i logiske enheder (discoverability).
- Hierarkiske, “.” separerer komponenter.
- Kan indlejres i andre namespaces.
- Kan referere til typer i et namespace ved:
 - **using**
 - fuld kvalificeret (namespace + typenavn).
- Altid et default (globalt) namespace, deklARATIONER udenfor eksplicitte namespaces havner her.

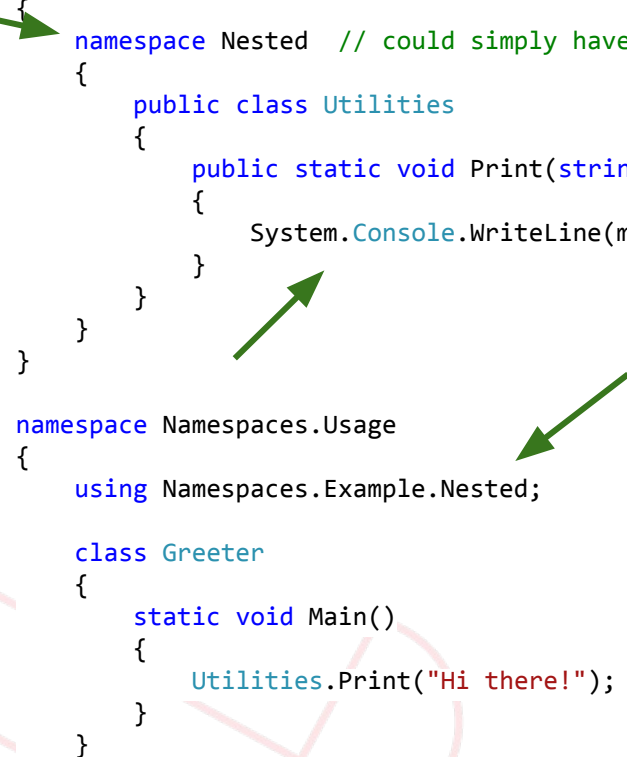
Kodestruktur - namespaces og using

Namespace hierarki og fuld kvalificeret

```
namespace Namespaces.Example
{
    namespace Nested // could simply have used Namespaces.Example.Nested above
    {
        public class Utilities
        {
            public static void Print(string message)
            {
                System.Console.WriteLine(message);
            }
        }
    }
}

namespace Namespaces.Usage
{
    using Namespaces.Example.Nested;

    class Greeter
    {
        static void Main()
        {
            Utilities.Print("Hi there!");
        }
    }
}
```



The diagram illustrates the namespace hierarchy and usage. A green arrow points from the `namespace Nested` line to the `using Namespaces.Example.Nested;` line in the `Namespaces.Usage` namespace, indicating the fully qualified path. Another green arrow points from the `Utilities` class in the `Nested` namespace to the `Utilities.Print` call in the `Main` method of the `Greeter` class, showing how the `using` statement allows for unqualified access to the `Utilities` class.

Kodestruktur - Klasser I

- Svarer til klasser i Java, C++.
- Definerer hvordan en datatype ser ud, men uden data - hvad en objekt instans af klassen indeholder.
- Erklæres med **class**.
- Instantieres med **new**.
- Kan bruge **this** i members til at tilgå den aktuelle instans.

Kodestruktur - Klasser II

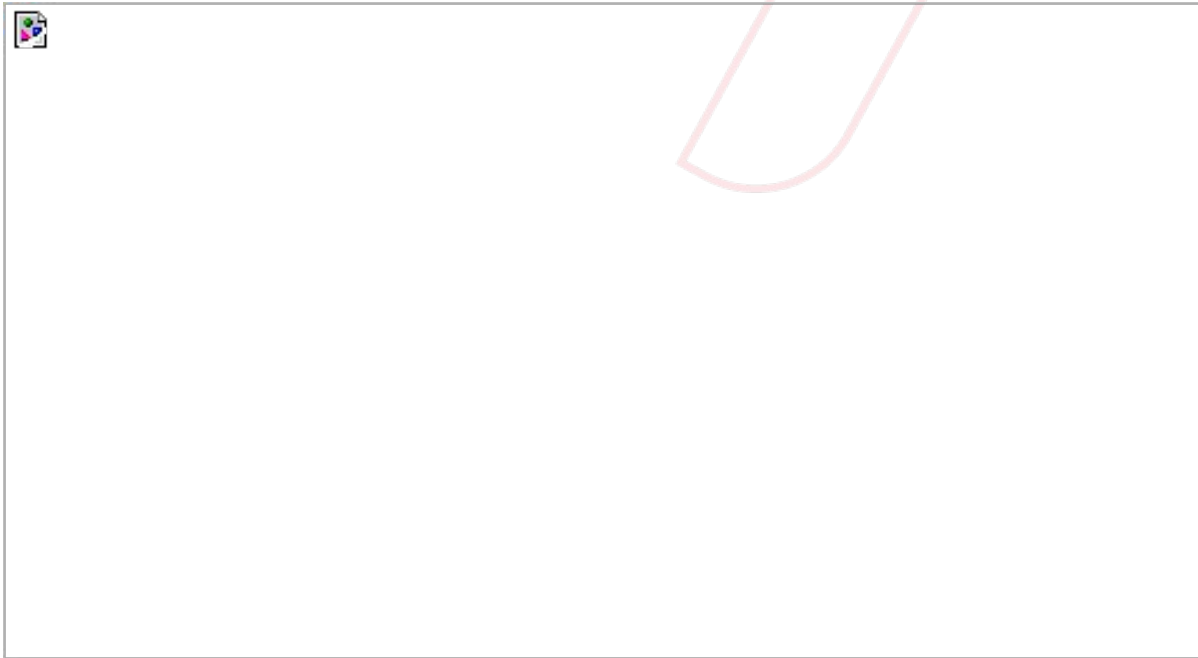
Definere og bruge en klasse

```
public class Vector2D
{
    public float x, y;
    public Vector2D(float x, float y)
    {
        this.x = x;
        this.y = y;
    }
}

public class Usage
{
    public void Main()
    {
        Vector2D v1 = new Vector2D(1, 0);
        Vector2D v2 = new Vector2D(0, 1);

        // vector addition
        v1.x += v2.x;
        v1.y += v2.y;

        System.Console.WriteLine("v1 = (" + v1.x + ", " + v1.y + ")");
    }
}
```

Kodestruktur - Felter

- Variabel i en klasse.
- Objektets data.
- Har en datatype.
- Bruges i metoder og properties.
- Regulere adgang via access modifiers.
- **static** felter: data på klasse i stedet for instans.
- **readonly** felter: kan ikke ændres efter initialisation.

```
public sealed class Coordinate
{
    public static readonly Coordinate Origin =
        new Coordinate(0, 0);

    private readonly int x;
    private readonly int y;

    public Coordinate(int x, int y)
    {
        this.x = x;
        this.y = y;
    }

    public int X { get { return this.x; } }

    public int Y { get { return this.y; } }
}
```

Kodestruktur - Metoder

- Funktion der:
 - har adgang til alle felter i klassen.
 - indeholder statements.
 - kan bruge **void** til at angive at den ikke returnerer værdi.
 - kan bruge **return** til at returnere fra metoden, evt. med værdi.
 - kan *overloades* - flere metoder kan have samme navn, hvis de har forskellige parametre.
- **C# 6.0: Expression body definitions** - kan angive metode med lambda i stedet for blok.
- *Avanceret: **yield return** kan bruges til at lave en iterator metode.*

```
class Rectangle
{
    private float length;
    private float height;

    float GetLength()
    {
        return length;
    }

    float GetArea() => length * height;

    void PrintArea(string pre, string post)
    {
        Console.Write(pre);
        Console.Write(this.GetArea());
        Console.WriteLine(post);
    }

    void PrintArea(string prefix)
    {
        PrintArea(prefix, "mm²");
    }

    void PrintArea()
    {
        PrintArea("Rectangle: ");
    }
}
```

Kodestruktur - Parametre

- Input værdier til metoden.
- Brug **params** for “varargs”:
 - Parameter typen skal være array.
 - Skal være sidst i parameterlisten.
 - Metode kan kaldes med vilkårligt antal argumenter på den plads.
- Optional arguments:
 - Brug `= expr` efter parameternavn.
 - Kalder behøver ikke udfylde.
- Named arguments:
 - Efter positional arguments.
 - Fri indbyrdes orden.
- *Avanceret: **ref** kan bruges til pass-by-reference og **out** kan bruges til multiple return.*

```
static int Add(int a, int b, params int[] rest)
{
    int result = a + b;
    foreach (int i in rest)
        result += i;
    return result;
}

// can take any number of extra arguments
int ten = Add(1, 2, 3, 4);
int four = Add(2, 2); // ...including none

static XmlReader Create(string uri,
    XmlReaderSettings settings = null,
    XmlParserContext context = null)
{
    // Create has 12 overloads!
    return XmlReader.Create(
        uri, settings, context);
}

XmlReader r1 = Create("http://ex.dk/xml");

XmlReader r2 = Create("http://ex.dk/xml", new
    XmlReaderSettings { IgnoreWhitespace = true });

XmlReader r3 = Create("http://ex.dk/xml",
    context: new XmlParserContext(null, null, null,
    XmlSpace.Preserve));
```

Kodestruktur - Constructors I

- Speciel metode der returnerer ny instans af klassen.
- Metodenavn = klassenavn.
- Initialiserer felter.
- Kan tage parametre.
- Kan have flere constructors med forskellige parametre.
- Hvis du ikke laver nogen, får klassen en default constructor uden parametre.

```
public class Die
{
    // No explicit constructors -> default added
    private int pips = new Random().Next(1, 7);
    public int GetRoll() { return pips; }
}

// Call the implicit constructor
Die die = new Die();

public class Shift
{
    private DateTime start, end;

    public Shift(DateTime start, DateTime end)
    {
        this.start = start;
        this.end = end;
    }

    public Shift(DateTime start, TimeSpan len)
        : this(start, start.Add(len)) { }
}
```

Kodestruktur - Constructors II

- Constructor chaining:
 - Kan kalde constructor fra basis klasse via “base”.
 - Kan kalde constructor fra egen klasse via “this”.
- Kan skjule constructor ved at erklære private.
- **C# 6.0: Primary constructors.**
- *Avanceret: static modifier angiver klasse-constructor - kan bruges til at initialisere statiske felter.*

```
public abstract class Animal {  
    private int numberOfLegs;  
  
    protected Animal(int numberOfLegs) {  
        this.numberOfLegs = numberOfLegs;  
    }  
}  
  
public class Millipede : Animal {  
    public Millipede() : base(1000) { }  
}  
  
public class Point {  
    private float x, y;  
  
    private Point(float x, float y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    public static Point At(float x, float y) {  
        return new Point(x, y);  
    }  
}  
  
Point p = Point.At(0.0f, 0.0f);
```

Kodestruktur - Properties

- Ligner felter udefra.
- Specielle **get** og **set** metoder kaldet *accessors*.
- Benytter **value** i setters til at tilgå argument.
- Auto-implemented properties - giver automatisk backing field.
- **C# 6.0:** Kan initialisere auto-implemented getter-only properties.
- **C# 6.0:** Expression body definition.

```
public class TimePeriod {
    private double seconds;

    public double Hours
    {
        get { return seconds / 3600; }
        set { seconds = value * 3600; }
    }

    public void Use()
    {
        TimePeriod t = new TimePeriod();
        t.Hours = 24; // 'set' accessor called

        // 'get' accessor called.
        Console.WriteLine("Hours: " + t.Hours);
    }
}

public class TimePeriod2 {
    public double Seconds { get; set; }
    public double Hours
    {
        get { return Seconds / 3600; }
        set { Seconds = value * 3600; }
    }
}
```

Kodestruktur - Events

- En C# event er:
 - Notifikation til klassens klienter om at noget interessant er sket.
 - Publish/subscribe pattern.
 - En form for callback.
- Bruger en *delegate* til at angive metode signatur for handlers.
 - Dem hører vi mere om i modul 4.
- Kan kun raises inde fra klassen.
- Klienter bruger `+=`/`-=` til at subscribe/unsubscribe.
- Bemærk: *nemt at komme til at lave memory-leaks!*
- Avanceret: kan lave egne accessors.

```
class EventArgs : EventArgs {  
    public Exception Error { get; private set; }  
    public EventArgs(Exception error) {  
        this.Error = error;  
    }  
}  
  
class Worker {  
    public event EventHandler<EventArgs> Error;  
  
    private void RaiseError(Exception error) {  
        var handler = Error;  
        if (handler != null)  
            handler(this, new EventArgs(error));  
    }  
}  
  
class Logger {  
    public Logger(Worker publisher) {  
        publisher.Error += HandleError;  
    }  
  
    private void HandleError(object sender,  
        EventArgs args) {  
        Console.WriteLine("Error from: " + sender);  
        Console.WriteLine(" - " + args.Error);  
    }  
}
```


Objekt Orienteret Programmering



OOP - Encapsulation I

- Access modifiers styrer adgang til klasser og deres members.
- C# definerer følgende modifiers:

public	Type eller member kan tilgås fra samme og andre assemblies.
private	Type eller member kan kun tilgås i samme klasse.
protected	Type eller member kan tilgås fra samme klasse og underklasser.
internal	Type eller member kan tilgås i kode fra samme assembly.
protected internal	Kan tilgås både som protected og som internal.

OOP - Encapsulation II

Indkapsle data i klasser

```
public class Vector2D
{
    private float x, y;

    public Vector2D(float x, float y)
    {
        this.x = x;
        this.y = y;
    }

    public void Add(Vector2D other)
    {
        this.x += other.x;
        this.y += other.y;
    }
}

public class Usage
{
    public void Main()
    {
        Vector2D v1 = new Vector2D(1, 0);
        Vector2D v2 = new Vector2D(0, 1);
        v1.Add(v2); // v1.x is now 1 and v1.y is 1
    }
}
```



OOP - Nedarvning

- Angives via : `SomeClass`.
- C# understøtter *ikke* multiple inheritance:
 - Kun én base class.
 - Kan implementere vilkårligt antal interfaces.
- **abstract** klasser:
 - Kan indeholde **abstract** members.
 - Kan ikke instantieres.
- **sealed** klasser:
 - Kan ikke nedarves fra.
- Brug **base** til at:
 - Vælge superklasse constructor.
 - Tilgå superklassens members.

```
abstract class Car {  
    public abstract void Drive();  
}  
  
abstract class ElectricCar : Car {  
    public override void Drive()  
    {  
        // Send electricity to the engine  
    }  
}  
  
sealed class Tesla : ElectricCar {  
    public override void Drive()  
    {  
        // Start logging road data  
  
        base.Drive(); // call ElectricCar.Drive  
    }  
}  
  
// Error: cannot instantiate abstract class  
// ElectricCar car = new ElectricCar();  
  
// Error: cannot inherit from sealed class  
// class Copycat : Tesla { }
```

OOP - Polymorfi

- “Polymorfiske modifiers”:
 - Disse kan bruges på metoder, properties, indexers og events.
- **abstract** members:
 - Underklasser ansvarlige for impl.
- **virtual** members:
 - Kan overstyre i underklasser.
- **sealed** members:
 - Underklasser kan ikke overstyre.
 - Er default.
- **override** modifier:
 - Overstyre superklasse member.
- **new** modifier:
 - Skjule superklasse member.

```
class Car {
    public virtual string Fuel {
        get { return "gasoline"; }
    }
}

class ElectricCar : Car {
    public sealed override string Fuel {
        get { return "electricity"; }
    }
}

class Tesla : ElectricCar {
    // Error: cannot override sealed method
    //public override string Fuel { get { return
    "Musk"; } }

    // ... but we can explicitly shadow it
    public new string Fuel {
        get { return "Musk"; }
    }
}

public void Use() {
    Tesla tesla = new Tesla();
    Console.WriteLine(tesla.Fuel); // Musk
    ElectricCar car = tesla;
    Console.WriteLine(car.Fuel); // electricity
}
```

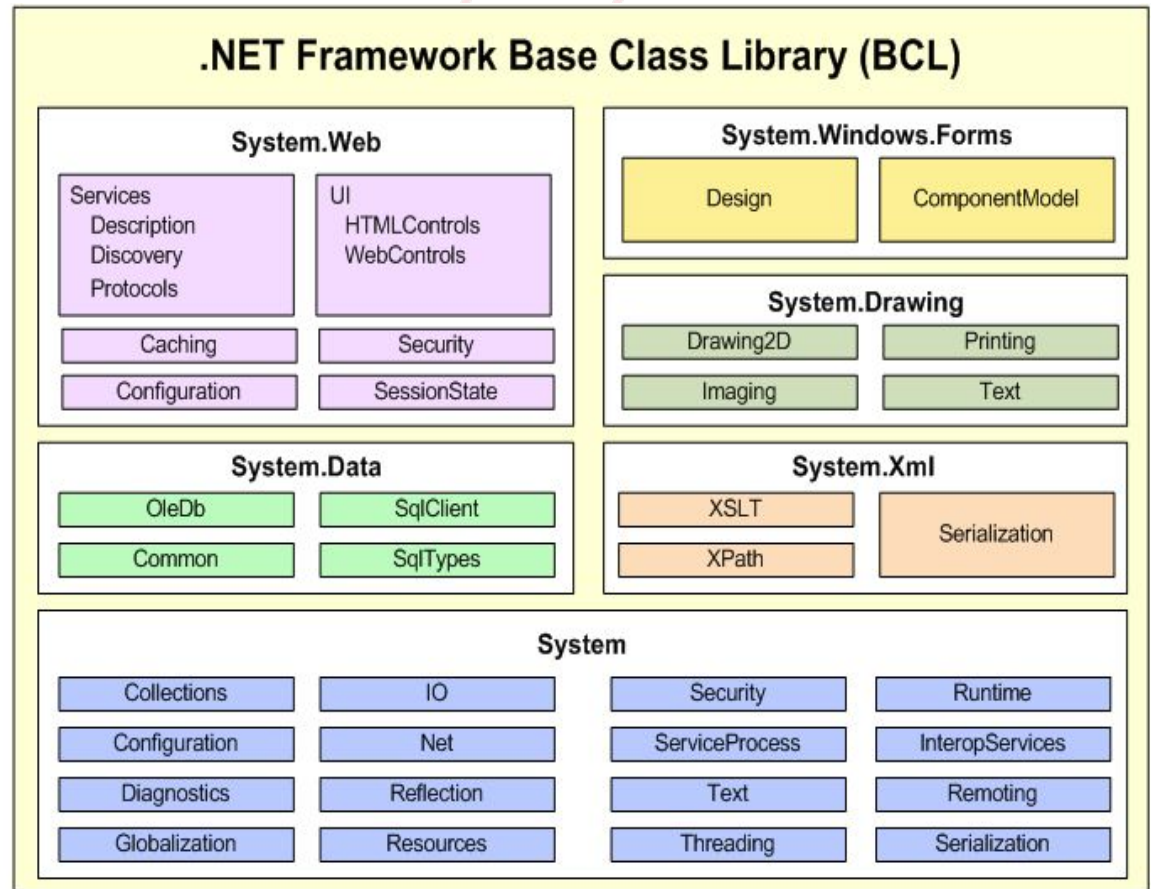
.NET Base Class Library

.NET - Type hierarki



.NET Base Class Library (BCL)

- BCL er et bibliotek med funktionalitet fælles for alle .NET sprog.
- Alle normale basis funktioner der bruges i et program er her.
- Inkluderer:
 - Stream handling (inkl. fil skrivning / læsning).
 - Grafik rendering
 - Tråd håndtering
 - XML manipulation
 - Globalisering
 - Basis diagnostics.



.NET Base Class Library (BCL)

- I .NET er typerne ens på tværs af sprog og platforme.

.NET Base Class Library (BCL)

- Alle unsigned typer nummer typer har en ækvivalent der er præfixet u, f.eks. unsigned short == ushort.
- Nullables gør livet nemmere.
- Alle value types kan gøres nullable ved at tilføje “?” efter type navnet:

```
static void Main()
{
    int? a = 6;
    int b = 0;
    if(a.HasValue)
    {
        b=a.Value;
    }
}
```

Data Type	Range
byte	0 .. 255
sbyte	-128 .. 127
short	-32,768 .. 32,767
ushort	0 .. 65,535
int	-2,147,483,648 .. 2,147,483,647
uint	0 .. 4,294,967,295
long	-9,223,372,036,854,775,808 .. 9,223,372,036,854,775,807
ulong	0 .. 18,446,744,073,709,551,615
float	-3.402823e38 .. 3.402823e38
double	-1.79769313486232e308 .. 1.79769313486232e308
decimal	-79228162514264337593543950335 .. 79228162514264337593543950335
char	A Unicode character.
string	A string of Unicode characters.
bool	True or False.
object	An object.

BCL - Collections

- Collections er en måde at holde styr på mange data.
- .NET base library indeholder flere collection classes.
 - ArrayList (alternative to array)
 - Hashtable (uses key to access values)
 - SortedList (key and index)
 - Stack (LiFo)
 - Queue (Fifo)

... og mange flere.

- Alle disse er objekttyper, og har typisk en Add (object) metode.

```
static void Main()
{
    int a = 6;
    string b = "snugglewoofie";

    ArrayList al = new ArrayList();
    al.Add(a);
    al.Add(b);
}
```

BCL - Generic Collections

- Generics er en måde at typesikre en metode eller collection.
- Generics er mere effektive end unchecked collections
- De to mest brugte er:
 - List<T>
 - Dictionary<TKey, TValue>
- I kommer til at bruge begge i Xamarin.

```
static void Main()
{
    int a = 6;
    string b = "snugglewoofie";

    List<int> integerList = new List<int>();
    integerList.Add(a);
    integerList.Add(b);
}
```

```
class Program
{
    0 references
    static void Main()
    {
        MyClass myClass = new MyClass();
        int a = 6;
        string b = "snugglewoofie";

        Dictionary<string, MyClass> myDictionary =
            new Dictionary<String, MyClass>();

        myDictionary.Add("myClassKey", myClass);
        myDictionary.Add("myIntegerKey", a);
        myDictionary.Add("myStringKey", b);
    }
}

4 references
class MyClass()
{
    int integerA;
    int integerB;
}
```

.NET Base Class Library (BCL)

Vigtige type fra **System**:

- Convert - konvertere data.
- Console - std. IO for text og error streams.
- Lazy<T> - først initialisere et objekt når du skal bruge det.
- Math - formler og udregninger
- Random - generere tilfældige tal
- Progress<T> - rapportere fremskridt asynkront
- Nullable
- Tuple - nem måde at returnere multiple værdier
- DateTime
- TimeSpan
- TimeZone



Referencer

- Statements:
 - [MSDN om statements](#)
- Parametre:
 - [Jon Skeet om parametre](#)
 - [Lee Richards om parametre](#)
- Events:
 - [Eric Lippert om events](#)
- Nyt i C# 6.0:
 - [Roslyn wiki](#)
- Kildekode for .NET:
 - [referencesource](#)