

Praksis eksempel Debugging i VS

Assemblies Github

Attributes

Dynamic programming

Multithreading

Grundlæggende programmering

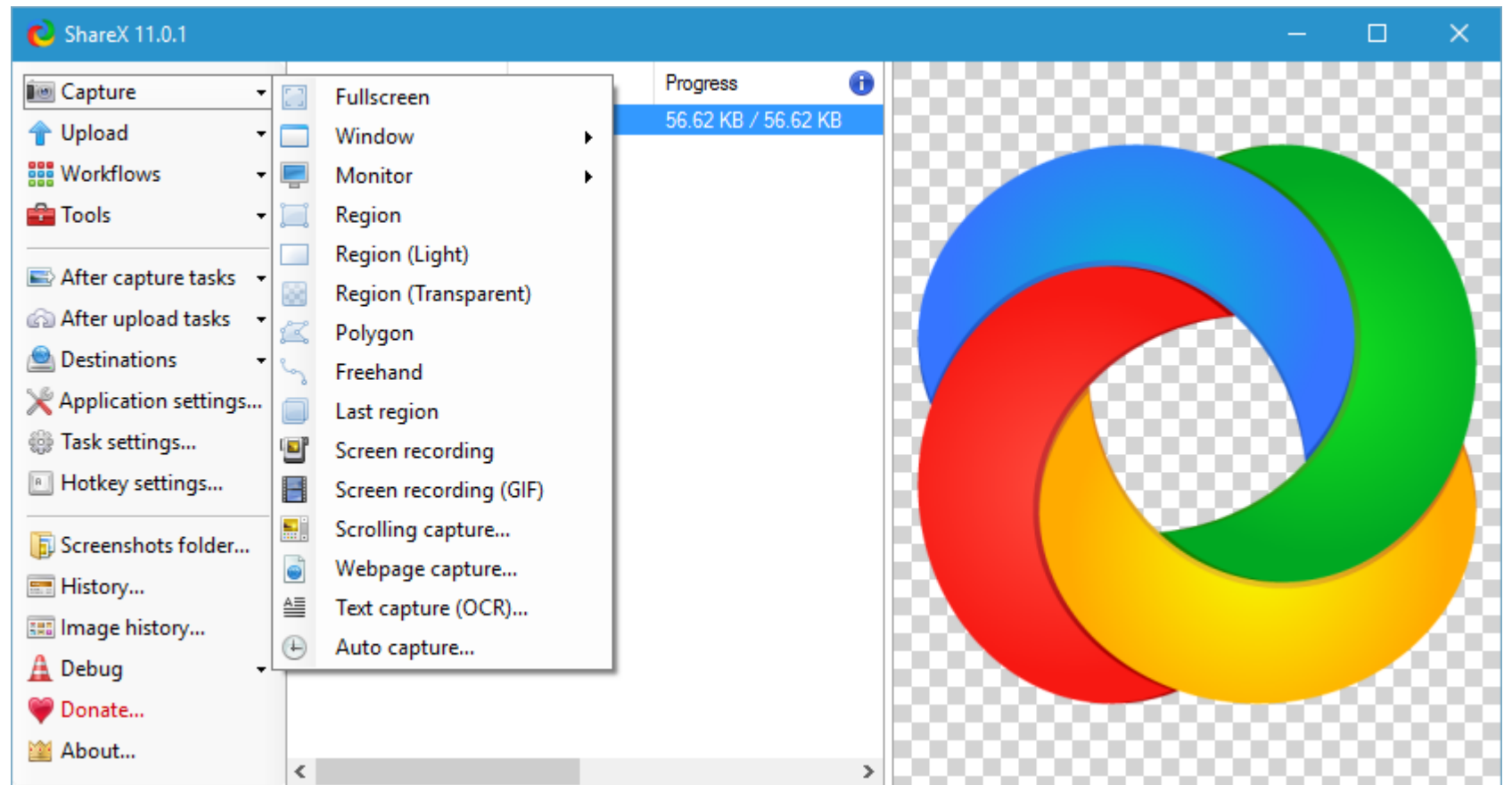
Lektion 6

Praksis eksempel

ShareX, et skærm-optager program

Praksis eksempel ShareX

- ShareX er et gratis open source program som lader en capture eller optage ethvert område af skærmen, og dele den med et enkelt tryk på en tast.
- Det giver også mulighed for at uploade billeder, tekst eller andre typer filer til mere end 80 understøttede destinationer.





Assemblies

Samlet kode



Assemblies

- Et assembly er en fil, der er automatisk genereres af compileren efter en vellykket samling af hver NET applikation.
- Det kan enten være et Dynamic Link Library eller en eksekverbar fil.
- Det genereres kun en gang for et program og ved hver efterfølgende kompilering bliver assembly opdateret.
- Hele processen vil køre i baggrunden af ens applikation.
- Et assembly indeholder Intermediate Language (IL) kode, som svarer til Java byte kode.
- I .NET sproget består det af metadata. Metadata enumerates features i hver "type" inde i assemblyet eller binary.

Assemblies

- Ud over metadata har assemblies også en særlig fil kaldet Manifest. Den indeholder oplysninger om den aktuelle version af assemblyet og andre relaterede oplysninger.
- I .NET, er der to slags assemblies: Single fil og Multi-fil.
- En enkelt fil assembly indeholder alle de nødvendige oplysninger (IL, Metadata, og Manifest) i en enkelt pakke.
- De fleste assemblies i .NET består af enkelt fil assemblies.
- Multi fil assemblies er sammensat af en lang række NET binære filer eller moduler og genereres til større applikationer. Et assembly vil indeholde et manifest og andre vil have IL og Metadata instruktioner.

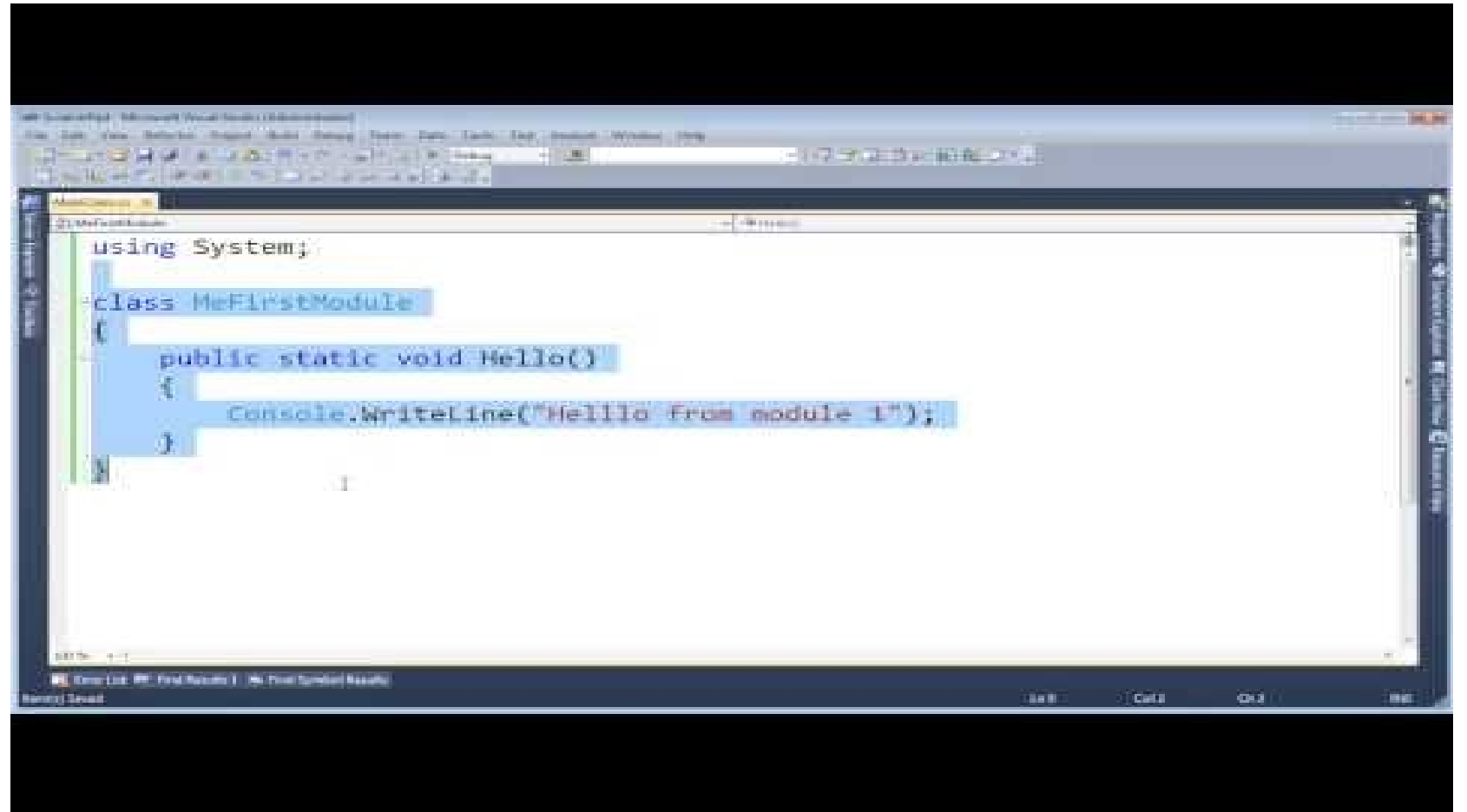
Assemblies

- Ud over metadata har assemblies også en særlig fil kaldet Manifest. Den indeholder oplysninger om den aktuelle version af assemblyet og andre relaterede oplysninger.
- I .NET, er der to slags assemblies: Single fil og Multi-fil.
- En enkelt fil assembly indeholder alle de nødvendige oplysninger (IL, Metadata, og Manifest) i en enkelt pakke.
- De fleste assemblies i .NET består af enkelt fil assemblies.
- Multi fil assemblies er sammensat af en lang række NET binære filer eller moduler og genereres til større applikationer. Et assembly vil indeholde et manifest og andre vil have IL og Metadata instruktioner.

Assemblies

- De fleste C# assemblies er som sagt EXE filer, men der er også library assemblies, der er DLL-filer.
- De omtales ofte som klasse biblioteker, fordi de indeholder klasser, som du kan bruge i dine programmer.
- Namespaces spiller en stor rolle i, hvordan man bruger dem.
- En nem måde at lave en library collection er at lave et Class Library projekt i Visual Studio ved at vælge "Class Library" som projekt type, når du opretter et nyt projekt.

Assemblies



```
using System;

class MeFirstModule
{
    public static void Hello()
    {
        Console.WriteLine("Hello from module 1");
    }
}
```



Attributes

Information om adfærd



Attributes

- En attribut er en deklarativ tag, der bruges til at formidle information til runtime om adfærd af forskellige elementer som klasser, metoder, strukturer, enumerators, samlinger etc. i et program.
- Man kan tilføje deklarativ information til et program ved hjælp af en attribut.
- Et deklarativt tag er vist ved firkantede ([]) parenteser placeret over elementet det anvendes til.
- Attributter bruges til at tilføje metadata, såsom compiler instruktion og andre oplysninger, såsom kommentarer, beskrivelse, metoder og klasser til et program.
- .NET Frameworket indeholder to typer af attributter: **de foruddefinerede attributter** (pre-defined) og **specialbyggede attributter** (custom built).

Attributes

- En attribut angives

```
[attribute(positional_parameters, name_parameter = value, ...)] element
```

- Navn på den attribut, og dens værdier er angivet i de firkantede parenteser, før det element, som attributten anvendes på.
- Positionelle parametre angiver de væsentlige oplysninger og navnet parametre angiver valgfri oplysninger.
- .Net Frameworket tilbyder tre prædefinerede attributer:
 - AttributeUsage
 - Conditional
 - Obsolete

Attributes

AttributeUsage

- Den præ-definerede attribut **AttributeUsage** beskriver hvordan en brugerdefineret attribut klasse kan bruges.
 - Den specificerer de typer af elementer, som attributten kan anvendes på.
- Syntaks for at specificere denne attribute

```
[AttributeUsage( validon, AllowMultiple=allowmultiple, Inherited=inherited )]
```

- Parameteren **validon** specificerer de sprog elementerne som attribut kan placeres på. Det er en kombination af værdien af en enumerator *AttributeTargets*. Standard værdien er *AttributeTargets.All*.
- Parameteren **allowmultiple** (valgfri) giver værdi til *AllowMultiple* property for denne attribut, en boolsk værdi. Hvis dette er sandt, er attributten Multiuse. Standard er falsk (engangsbrug).

Attributes AttributeUsage

- Parameteren `inherited` (valgfri) giver værdi til *Inherited* property for denne attribut, en Boolean værdi. Hvis det er sandt, bliver attributten arvet af afledte klasser. Standardværdien er false (ikke ned-arvet).
- Et samlet eksempel:

```
[AttributeUsage(AttributeTargets.Class |  
AttributeTargets.Constructor |  
AttributeTargets.Field |  
AttributeTargets.Method |  
AttributeTargets.Property, AllowMultiple = true)]
```

Attributes

AttributeUsage

Conditional

- Denne prædefinerede attribut markerer en conditional method hvis udførelse afhænger af en angivet preprocessing identifier.
- Det forårsager betinget kompilering af metodekald, afhængigt af den angivne værdi, såsom **Debug** eller **Trace**. For eksempel, viser værdierne af variablerne mens en kode debugges.

- Syntaks

```
[Conditional( conditionalSymbol )]
```

- For eksempel

```
[Conditional("DEBUG")]
```

Attributes

AttributeUsage

Conditional

```
1 using System;
2 using System.Diagnostics;
3
4 public class Myclass
5 {
6     [Conditional("DEBUG")]
7     public static void Message(string msg)
8     {
9         Console.WriteLine(msg);
10    }
11 }
12
13 class Test
14 {
15     static void function1()
16     {
17         Myclass.Message("In Function 1.");
18         function2();
19     }
20     static void function2()
21     {
22         Myclass.Message("In Function 2.");
23     }
24
25     public static void Main()
26     {
27         Myclass.Message("In Main function.");
28         function1();
29         Console.ReadKey();
30     }
31 }
```


Attributes

AttributeUsage

Obsolete

- Denne foruddefineret attribut markerer en program enhed, som ikke bør anvendes.
- Det giver dig mulighed for at informere compileren om at kassere et bestemt mål element.
 - For eksempel, når der anvendes en ny metode i en klasse, og hvis du stadig ønsker at beholde den gamle metode i klassen, kan du markere den som forældet ved at vise en meddelelse om at den nye metode bør anvendes i stedet for den gamle metode.
- Syntaks
 - [**Obsolete**(message)]
 - [**Obsolete**(message, iserror)]
- Hvor
 - Parameteren *message* er en streng, der beskriver grunden til at elementet er forældet og hvad der skal bruges i stedet.
 - Parameteren *iserror* er en boolesk værdi. Hvis dens værdi er sand, skal compileren behandle brugen af elementet som en fejl. Standardværdien er falsk (compiler genererer en advarsel).

Attributes

AttributeUsage

Obsolete

```
1 using System;
2
3 0 references
4 public class MyClass
5 {
6     1 reference
7     [Obsolete("Don't use OldMethod, use NewMethod instead", true)]
8     static void OldMethod()
9     {
10         Console.WriteLine("It is the old method");
11     }
12     0 references
13     static void NewMethod()
14     {
15         Console.WriteLine("It is the new method");
16     }
17     0 references
18     public static void Main()
19     {
20         OldMethod();
21     }
22 }
```

Attributes

Custom Attributes

- .NET frameworket tillader oprettelse af brugerdefinerede attributter, der kan bruges til at lagre deklarative oplysninger og kan hentes ved run-time. Denne information kan relateres til ethvert mål element afhængigt af designkriterier og program behov.
- Oprettelse og brug af brugerdefinerede attributter involverer fire trin:
 - Erklære en brugerdefineret attribut
 - Konstruktion af den brugerdefinerede attribut
 - Tilføj den brugerdefinerede attribut på et mål program element
 - Adgang attributter gennem refleksion
- Det sidste trin omfatter det at skrive et simpelt program til at læse igennem metadata til at finde forskellige notationer. Metadata er data om data eller oplysninger, der anvendes til at beskrive andre data. Dette program bør bruge refleksioner til at få adgang attributter ved runtime.

Attributes

Custom Attributes

- En ny brugerdefineret attribute bør være afledt af `System.Attribute` klassen.
- Herunder laver vi en brugerdefineret attribut kaldet `DebugInfo`

```
//a custom attribute BugFix to be assigned to a class and  
its members  
[AttributeUsage(AttributeTargets.Class |  
AttributeTargets.Constructor |  
AttributeTargets.Field |  
AttributeTargets.Method |  
AttributeTargets.Property,  
AllowMultiple = true)]  
  
public class DebugInfo : System.Attribute
```

Attributes

Custom Attributes

- Vi arbejder videre med `DeBugInfo` og lader den:
 - Kode nummeret for bug
 - Navnet på udvikleren der fandt bug
 - Dato for det sidste kode review
 - En streng besked til opbevaring af udviklerens bemærkninger
- `DeBugInfo` klassen har tre private properties til at gemme de første tre informationer og en public property til at gemme beskeden.
 - Derfor er bug number, udviklerens navn og dato for review lavet som positional parameters til `DeBugInfo` klassen og beskeden er en valgfri eller navngivet parameter.
- Hver attribute skal have mindst én constructor.
 - De positionelle parametre bør ledes gennem constructor.
- Den udvidede `DeBugInfo` klasse kan ses på næste slide.

Attributes

Custom Attributes

```
//a custom attribute BugFix to be assigned to a class and its members  
[AttributeUsage(AttributeTargets.Class |  
AttributeTargets.Constructor |  
AttributeTargets.Field |  
AttributeTargets.Method |  
AttributeTargets.Property, AllowMultiple = true)]
```

```
public class DeBugInfo : System.Attribute  
{  
    private int bugNo;  
    private string developer;  
    private string lastReview;  
    public string message;  
    public DeBugInfo(int bg, string dev, string d)  
    {  
        this.bugNo = bg;  
        this.developer = dev;  
        this.lastReview = d;  
    }  
    public int BugNo  
    {  
        get  
        {  
            return bugNo;  
        }  
    }  
}
```

Attributes

Custom Attributes

```
public string Developer
{
    get
    {
        return developer;
    }
}

public string LastReview
{
    get
    {
        return lastReview;
    }
}

public string Message
{
    get
    {
        return message;
    }
    set
    {
        message = value;
    }
}
}
```

Attributes

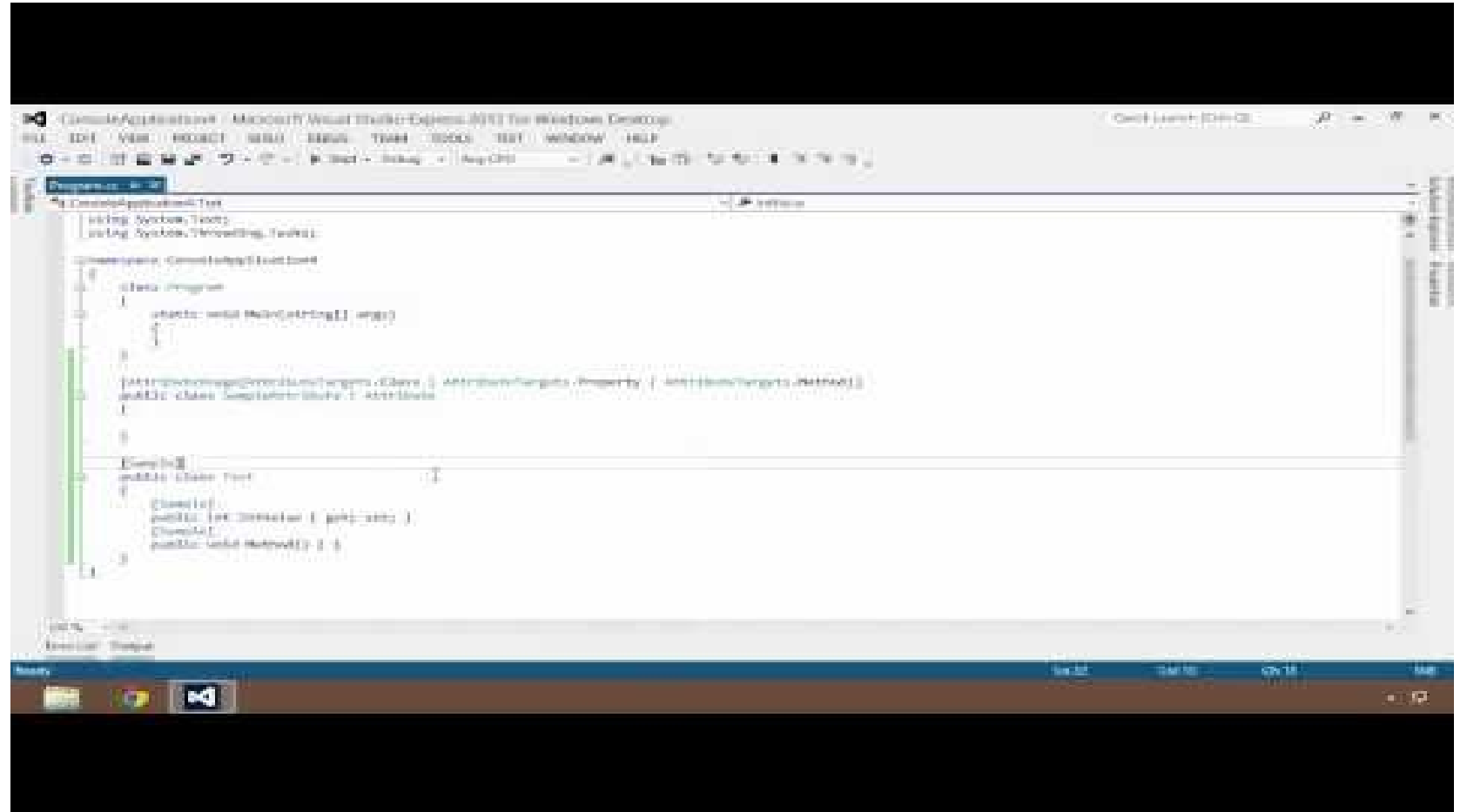
Custom Attributes

- Attributten anvendes ved at placere det umiddelbart før sit mål:

```
[DebuggerDisplay("Zara Ali, 12/8/2012, Message = Return type mismatch")]
[DebuggerDisplay("Nuha Ali, 10/10/2012, Message = Unused variable")]
class Rectangle
{
    //member variable
    protected double length;
    protected double width;
    public Rectangle(double l, double w)
    {
        length = l;
        width = w;
    }
    [DebuggerDisplay("Zara Ali, 19/10/2012, Message = Return type mismatch")]
    public double GetArea()
    {
        return length * width;
    }
    [DebuggerDisplay("Zara Ali, 19/10/2012")]
    public void Display()
    {
        Console.WriteLine("Length: {0}", length);
        Console.WriteLine("Width: {0}", width);
        Console.WriteLine("Area: {0}", GetArea());
    }
}
```


Attributes

Custom Attributes



```
using System;
using System.Reflection;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            [AttributeUsage(AttributeTargets.Class | AttributeTargets.Property | AttributeTargets.Method)]
            public class CustomAttribute : Attribute
            {
            }

            [Custom]
            public class Test
            {
                Custom();
                public int Summarize ( int i, int j );
                Custom();
                public void Method() { }
            }
        }
    }
}
```



Dynamic Programming

Refleksioner over indhold

Dynamic Programming Reflections

- **Reflection** objekter anvendes til opnåelse af type information ved kørelstidspunktet. De klasser, der giver adgang til metadata i et kørende program er i **System.Reflection** namespace.
- **System.Reflection** namespace indeholder klasser, der tillader en at få oplysninger om programmet og til dynamisk at tilføje typer, værdier og objekter til applikationen.
- Reflection har de følgende brugsmønstre:
 - Det giver view attribut information ved runtime.
 - Det gør det muligt at undersøge forskellige typer i et assembly og at instantiere disse typer.
 - Den tillader sen binding til metoder og egenskaber
 - Det gør det muligt at skabe nye typer ved runtime og derefter udfører nogle opgaver ved hjælp af disse typer.

Dynamic Programming Reflections Viewing metadata

- **MemberInfo** objektet fra **System.Reflection** klassen skal initialiseres for at opdage attributter knyttet til en klasse.
 - For at gøre dette definerer man et objekt af mål klassen, som:

```
System.Reflection.MemberInfo info = typeof(MyClass);
```

```
1  using System;
2
3  [AttributeUsage(AttributeTargets.All)]
   2 references
4  public class HelpAttribute : System.Attribute
5  {
6      public readonly string Url;
7
8      0 references
9      public string Topic // Topic is a named parameter
10     {
11         get
12         {
13             return topic;
14         }
15         set
16         {
17             topic = value;
18         }
19     }
```

Dynamic Programming Reflections

Viewing metadata

```
20 public HelpAttribute(string url) // url is a positional parameter
21 {
22     this.Url = url;
23 }
24 private string topic;
25 }
26
27 [HelpAttribute("Information on the class MyClass")]
28 class MyClass
29 {
30 }
31 namespace AttributeAppl
32 {
33     class Program
34     {
35         static void Main(string[] args)
36         {
37             System.Reflection.MemberInfo info = typeof(MyClass);
38             object[] attributes = info.GetCustomAttributes(true);
39             for (int i = 0; i < attributes.Length; i++)
40             {
41                 System.Console.WriteLine(attributes[i]);
42             }
43
44             Console.ReadKey();
45         }
46     }
47 }
```

Dynamic Programming Reflections

- Hent reflections.cs fra Fronter, den bygger videre på *DeBugInfo* eksemplet fra tidligere.

Dynamic Programming Reflections

```
using System;
using System.Linq;
using System.Reflection;

class Program
{
    static void Main()
    {
        Console.WriteLine("Types: " + Type.GetType(""));

        var props = Type.GetType("").GetProperties();
        foreach (var prop in props)
        {
            Console.WriteLine("Property: " + prop.Name);
        }

        var fields = Type.GetType("").GetFields();
        foreach (var field in fields)
        {
            Console.WriteLine("Field: " + field.Name);
        }

        var methods = Type.GetType("").GetMethods();
        foreach (var method in methods)
        {
            Console.WriteLine("Method: " + method.Name);
        }
    }
}

public class Sample
{
    public string Name { get; set; }
    public int Age { get; set; }
    public void Greet()
    {
    }
}
```

```
C:\Windows\system32\cmd.exe
C:\Windows\system32\cmd.exe /c powershell -Command "Get-Childitem -Path C:\Windows\system32\cmd.exe | Format-Table Name, CommandType, System.Object"
Type: Program
Property: Name
Property: Age
Property: Greet
Field: Name
Field: Age
Method: Greet
Press any key to continue . . .
```



Multithreading

Flow af program udførelse

Multithreading

- En tråd er defineret som udførelsesvejen af et program.
- Hver tråd definerer en unik strøm af kontrol.
 - Hvis en applikation indebærer komplicerede og tidskrævende operationer, så er det ofte nyttigt at indstille forskellige udførelses stier eller tråde, hvor hver tråd udfører et bestemt job.
- En tråd kaldes også en letvægts proces (**lightweight process**).
- Et almindeligt eksempel på brug af tråde er gennemførelsen af parallel programmering af moderne operativsystemer.
- Anvendelse af tråde sparer spild af CPU cyklus og øge effektiviteten af en applikation.

Programmering af tråde

En tråds livs cyklus

- En tråds livscyklus starter, når et objekt med `System.Threading.Thread` klassen skabes og slutter, når tråden er afsluttet eller fuldført.
- Følgende er de forskellige stater i en tråds livscyklus:
 - **The Unstarted State**: Det er den situation, hvor en forekomst af tråden er oprettet, men `Start` metoden er ikke kaldt.
 - **The Ready State**: Det er den situation, hvor tråden er klar til at køre og venter på CPU cyklus.
 - **The Not Runnable State**: En tråd er ikke eksekverbar når:
 - `Sleep` metoden er kaldt.
 - `Wait` metoden er kaldt.
 - Blokeret af I/O operationer.
 - **The Dead State**: Det er den situation, hvor tråden fuldender udførelse eller afbrydes.

Programmering af tråde Main thread

- I C # bliver `System.Threading.Thread` klassen brugt til at arbejde med tråde.
- Det gør det muligt at skabe og adgang til de enkelte tråde i en flertrådede anmodninger.
- Den første tråd der udføres i en proces kaldes den røde tråd.
- Når et C # program starter udførelse bliver den røde tråd oprettet automatisk. Trådene oprettet ved hjælp af `Thread` klassen kaldes barne-tråde af den røde tråd.
- Du kan få adgang en tråd ved hjælp af `CurrentThread` property af `Thread` klassen.

Programmering af tråde Main thread

```
1  using System;
2  using System.Threading;
3
4  namespace MultithreadingApplication
5  {
6      0 references
7      class MainThreadProgram
8      {
9          0 references
10         static void Main(string[] args)
11         {
12             Thread th = Thread.CurrentThread;
13             th.Name = "MainThread";
14             Console.WriteLine("This is {0}", th.Name);
15             Console.ReadKey();
16         }
17     }
18 }
```

Programmering af tråde

Egenskaber og metoder for Main thread

Property	Description
CurrentContext	Gets the current context in which the thread is executing.
CurrentCulture	Gets or sets the culture for the current thread.
CurrentPrinciple	Gets or sets the thread's current principal (for role-based security).
CurrentThread	Gets the currently running thread.
CurrentUICulture	Gets or sets the current culture used by the Resource Manager to look up culture-specific resources at run-time.
ExecutionContext	Gets an ExecutionContext object that contains information about the various contexts of the current thread.
IsAlive	Gets a value indicating the execution status of the current thread.

Programmering af tråde

Egenskaber og metoder for Main thread

IsBackground	Gets or sets a value indicating whether or not a thread is a background thread.
IsThreadPoolThread	Gets a value indicating whether or not a thread belongs to the managed thread pool.
ManagedThreadId	Gets a unique identifier for the current managed thread.
Name	Gets or sets the name of the thread.
Priority	Gets or sets a value indicating the scheduling priority of a thread.
ThreadState	Gets a value containing the states of the current thread.

Programmering af tråde

- Tråde laves ved at udvide Thread klassen. Den udvidede Thread klasse kalder derefter `Start()` metoden til at begynde barnetrådens udførelse.

```
1  using System;
2  using System.Threading;
3
4  namespace MultithreadingApplication
5  {
6      0 references
7      class ThreadCreationProgram
8      {
9          1 reference
10         public static void CallToChildThread()
11         {
12             Console.WriteLine("Child thread starts");
13         }
14
15         0 references
16         static void Main(string[] args)
17         {
18             ThreadStart childref = new ThreadStart(CallToChildThread);
19             Console.WriteLine("In Main: Creating the Child thread");
20             Thread childThread = new Thread(childref);
21             childThread.Start();
22             Console.ReadKey();
23         }
24     }
25 }
```

Programmering af tråde Administration af tråde

- Thread klassen giver forskellige metoder til håndtering af tråde.

Det følgende eksempel demonstrerer anvendelsen af `sleep()` fremgangsmåden til fremstilling af en tråd pause i en bestemt tidsperiode.

```
1  using System;
2  using System.Threading;
3
4  namespace MultithreadingApplication
5  {
6      0 references
7      class ThreadCreationProgram
8      {
9          1 reference
10         public static void CallToChildThread()
11         {
12             Console.WriteLine("Child thread starts");
13
14             // the thread is paused for 5000 milliseconds
15             int sleepfor = 5000;
16
17             Console.WriteLine("Child Thread Paused for {0} seconds", sleepfor / 1000);
18             Thread.Sleep(sleepfor);
19             Console.WriteLine("Child thread resumes");
20         }
21
22         0 references
23         static void Main(string[] args)
24         {
25             ThreadStart childref = new ThreadStart(CallToChildThread);
26             Console.WriteLine("In Main: Creating the Child thread");
27             Thread childThread = new Thread(childref);
28             childThread.Start();
29             Console.ReadKey();
30         }
31     }
32 }
```


Programmering af tråde Ødelæggelse af tråde

- `Abort()` metoden bruges til at ødelægge tråde.

Runtime afbryder tråden ved at kaste en `ThreadAbortException`. Denne undtagelse kan ikke blive fanget, styringen sendes til en *finally* blok, hvis nogen.

```
1  using System;
2  using System.Threading;
3
4  namespace MultithreadingApplication
5  {
6      class ThreadCreationProgram
7      {
8          public static void CallToChildThread()
9          {
10             try
11             {
12                 Console.WriteLine("Child thread starts");
13
14                 // do some work, like counting to 10
15                 for (int counter = 0; counter <= 10; counter++)
16                 {
17                     Thread.Sleep(500);
18                     Console.WriteLine(counter);
19                 }
20
21                 Console.WriteLine("Child Thread Completed");
22             }
23         }
24     }
25 }
```

Programmering af tråde Ødelæggelse af tråde

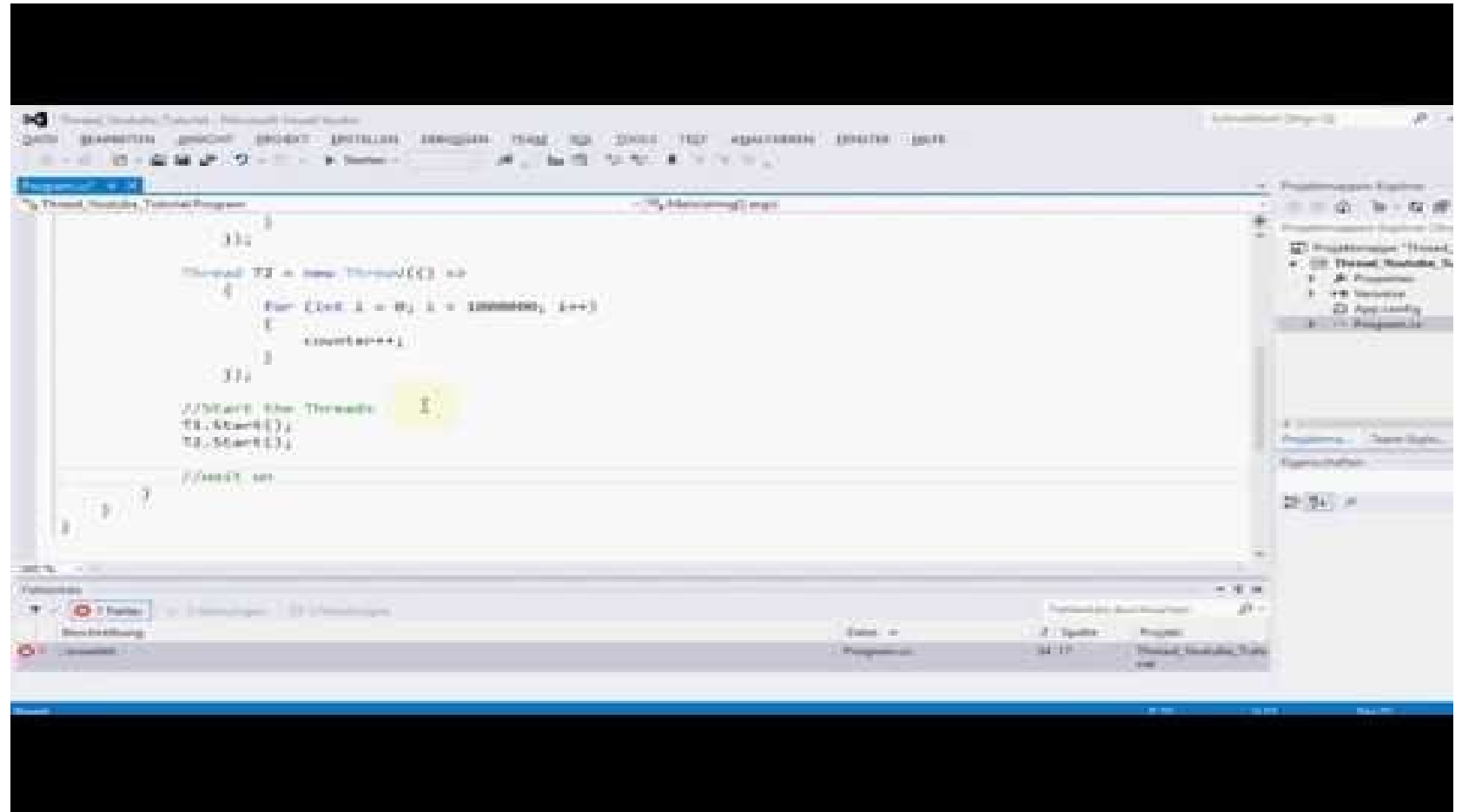
```
23
24     catch (ThreadAbortException e)
25     {
26         Console.WriteLine("Thread Abort Exception");
27     }
28     finally
29     {
30         Console.WriteLine("Couldn't catch the Thread Exception");
31     }
32 }
33
34 0 references
35 static void Main(string[] args)
36 {
37     ThreadStart childref = new ThreadStart(CallToChildThread);
38     Console.WriteLine("In Main: Creating the Child thread");
39     Thread childThread = new Thread(childref);
40     childThread.Start();
41
42     //stop the main thread for some time
43     Thread.Sleep(2000);
44
45     //now abort the child
46     Console.WriteLine("In Main: Aborting the Child thread");
47
48     childThread.Abort();
49     Console.ReadKey();
50 }
51 }
```

Programmering af tråde

Mutual exclusion

- I datalogi er gensidig udelukkelse (**mutual exclusion**) en egenskab ved concurrency kontrol, som er lavet med henblik på at forebygge race condition / hazard.
 - Race condition er når output er afhængig af sekvensen eller timingen af andre ukontrollerbare begivenheder.
- Mutual exclusion er kravet om, at en udførelses-tråd aldrig når sin kritiske strækning på samme tid som en anden samtidig tråd udførelse går ind i sin egen kritiske sektion.

Programmering af tråde



```
1 public class ThreadExample {
2     public static void main(String[] args) {
3         // Create a new Thread object
4         Thread T1 = new Thread() {
5             public void run() {
6                 for (int i = 0; i < 100000; i++)
7                     counter++;
8             }
9         };
10        // Start the Thread
11        T1.start();
12        // Print the counter value
13        System.out.println(counter);
14    }
15}
```



Debugging i Visual Studio

At tjekke ens program igennem



Debugging i VS

- Visual Studio debugger hjælper en med at observere run-time opførsel af ens program og finde problemer.
- Debuggeren fungerer med alle Visual Studio programmeringssprog og deres tilknyttede biblioteker.
- Med debugger kan man afbryde udførelsen af ens program for at undersøge ens kode, undersøge og redigere variabler, se registre, se instruktioner oprettet fra ens kildekode, og se hukommelsesplads brugt af ens program.

Debugging iVS



Visual Studio 2015



GitHub

Versionsstyring og deling af kode



Github

LEARN
github
in 20 minutes!

Opgave

Jeg står til rådighed til svar på spørgsmål og forklaring af fejl

Opgave

OPGAVE

- Begynd at tage prøverne inde på <https://mva.microsoft.com/en-US/training-courses/c-fundamentals-for-absolute-beginners-16169> fra en ende af.
- I behøves ikke se videoerne, men brug dem endelig som reference / genopfriskning.

Kilder

Materiale benyttet i denne lektion
Noget af det er udover pensum-listen!

Praksis eksempel

- <https://getsharex.com/>
- <https://github.com/ShareX/ShareX>

Assemblies

- http://www.codeguru.com/columns/csharp_learning/article.php/c5845/C-FAQ-15--What-is-an-Assembly.htm
- <http://broadcast.oreilly.com/2010/07/understanding-c-namespaces-and.html>
- https://youtu.be/ky3_3ubDxII

Attributes

- https://www.tutorialspoint.com/csharp/csharp_attributes.htm
- <https://youtu.be/mGkO7oM5VI8>

Dynamic programming

- https://www.tutorialspoint.com/csharp/csharp_reflection.htm
- [https://msdn.microsoft.com/en-us/library/hh156524\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/hh156524(v=vs.110).aspx)
- <https://youtu.be/3FvT6uNMT7M> (Reflection)

Multithreading

- https://www.tutorialspoint.com/csharp/csharp_multithreading.htm
- [https://msdn.microsoft.com/en-us/library/aa645740\(v=vs.71\).aspx](https://msdn.microsoft.com/en-us/library/aa645740(v=vs.71).aspx)
- <https://youtu.be/bxJzqNCZsNw> (Mutex)

Debugging i Visual Studio

- <https://msdn.microsoft.com/en-us/library/sc65sadd.aspx>
- <https://msdn.microsoft.com/en-us/library/y74odgd3.aspx>

Debugging i Visual Studio

- <https://msdn.microsoft.com/en-us/library/mt243867.aspx>
- <https://youtu.be/FHnxx9gDudI>

Github

- <https://www.pluralsight.com/blog/software-development/github-tutorial>
- <https://youtu.be/ofKg7e37bQE>