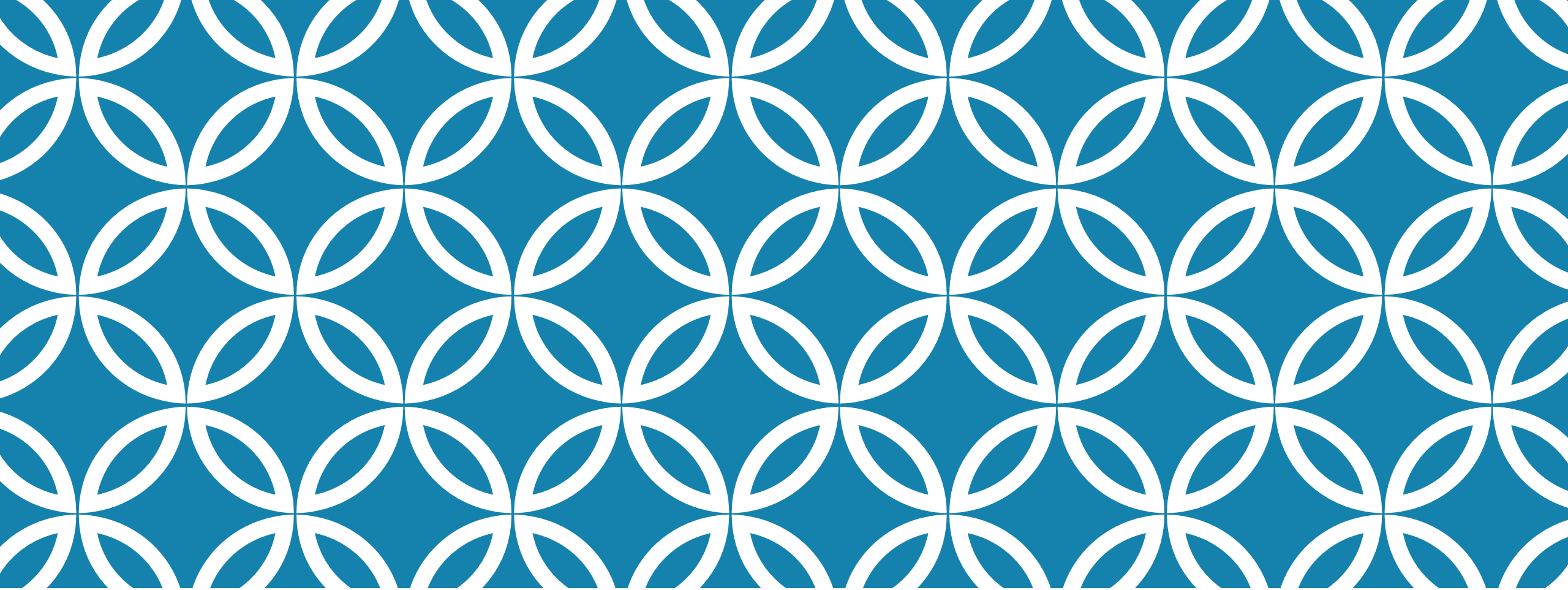
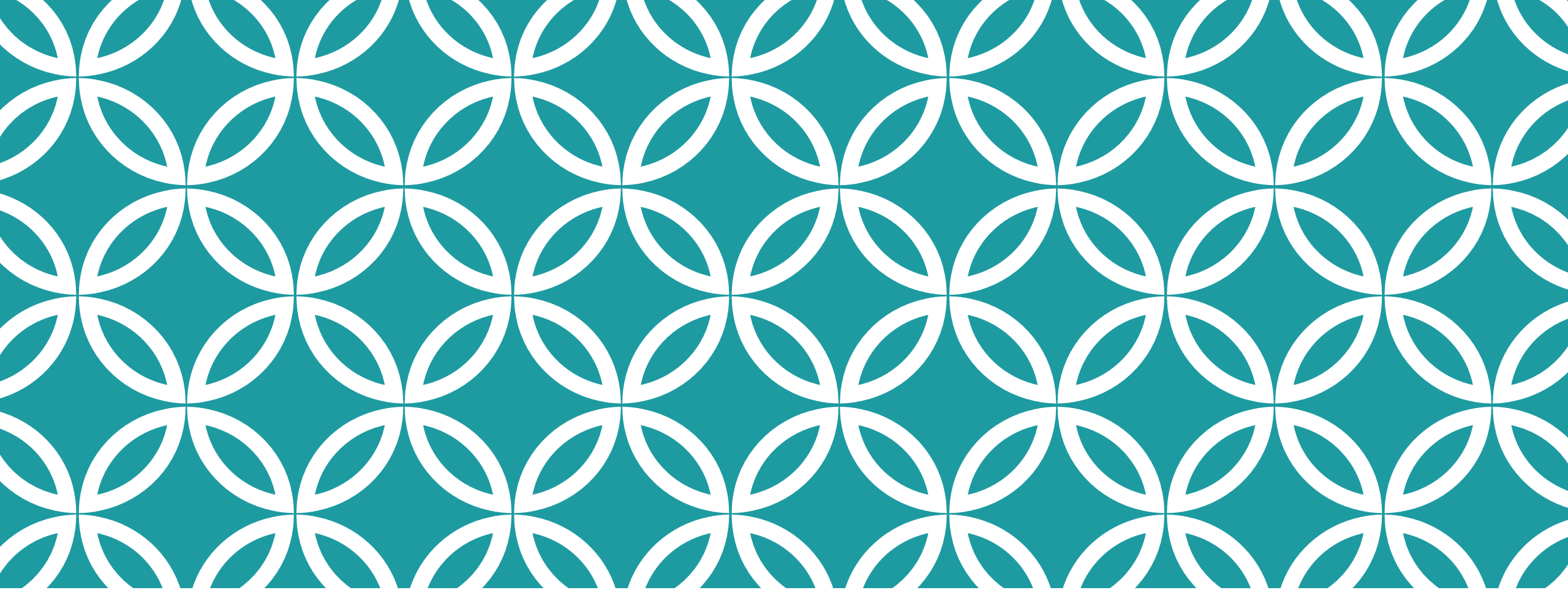


LØKKER METODER

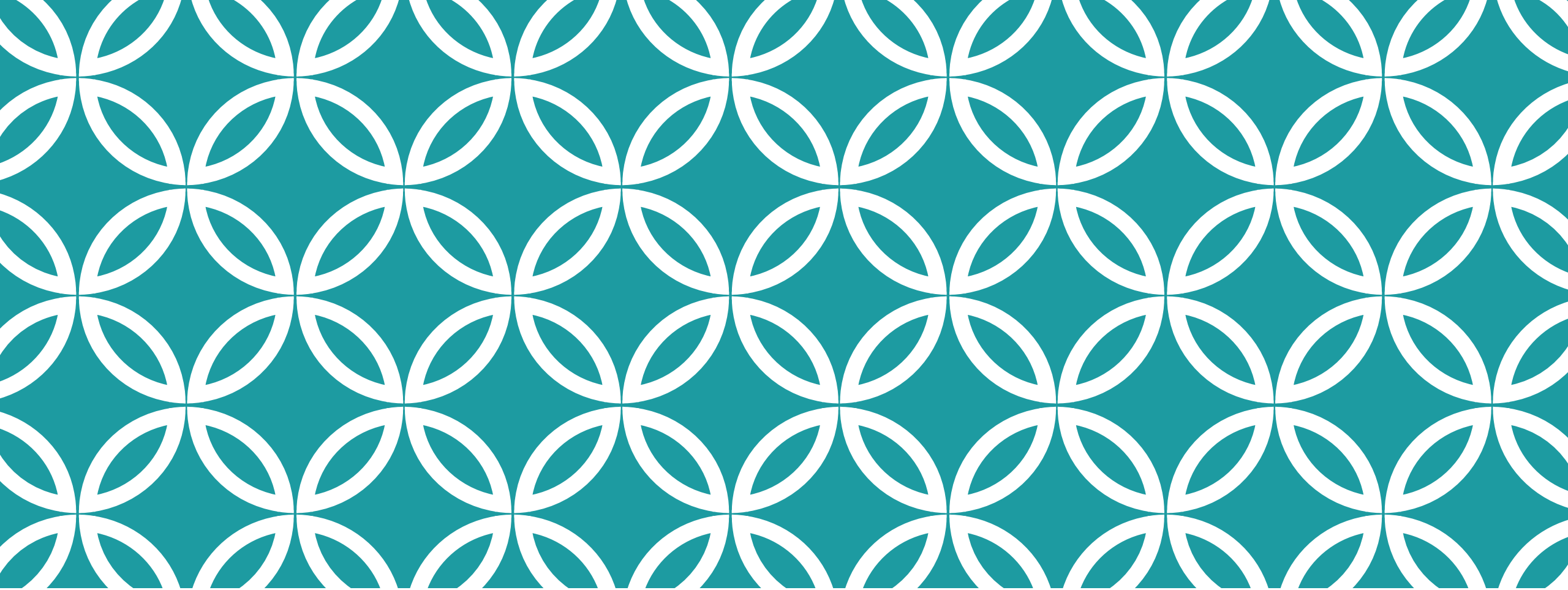
C'S HISTORIE

Grundlæggende programmering
Lektion 4



LEKTIER

UML struktur diagrammer om biler



OPERATORER

Syntaks til at udføre forskellige beregninger og handlinger

Booleans og hvorfor de er essentielle og praktiske

REPETITION

- Aritmetiske operatorer
  - Unary (påvirker ét element)
  - Binary (påvirker to elementer)
- Flow control
  - If sætninger

OPERATORER FLOW KONTROL

Code blocks

Code blocks kaldes ofte “scopes”. Scopes bruges til at afgøre, hvad ting et navn refererer til, et declaration space bestemmer, hvornår to ting erklæret med det samme navngive går i konflikt med hinanden.

Værdier er ikke tilgængelige uden for scope.

OPERATORER BINÆRE FLOW KONTROL

Relational og equality operators

Relationelle og ligestillings operators afgør om en værdi er større
end, mindre end eller lig med en anden værdi.

C # syntaks for ligestilling bruger ==, ligesom mange andre programmering sprog gør. For at afgøre om input lig 9, bruger man input == 9 operatoren for at adskille det fra tildelings operatoren, =.

Udråbstegnet betyder *ikke* i C #, så til test for ulighed du bruger ulighed operatoren ! =.

OPERATORER BINÆRE FLOW KONTROL

Relational og equality operators

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

OPERATORER BINÆRE

Relational og equality operators

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication6
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            int a = 21;
            int b = 10;

            if (a == b)
            {
                Console.WriteLine("Line 1 - a is equal to b");
            }
            else
            {
                Console.WriteLine("Line 1 - a is not equal to b");
            }

            if (a < b)
            {
                Console.WriteLine("Line 2 - a is less than b");
            }
            else
            {
                Console.WriteLine("Line 2 - a is not less than b");
            }

            if (a > b)
            {
                Console.WriteLine("Line 3 - a is greater than b");
            }
            else
            {
                Console.WriteLine("Line 3 - a is not greater than b");
            }
            /* Lets change value of a and b */
            a = 5;
            b = 20;

            if (a <= b)
            {
                Console.WriteLine("Line 4 - a is either less than or equal to  b");
            }

            if (b >= a)
            {
                Console.WriteLine("Line 5-b is either greater than or equal to b");
            }
            System.Console.ReadLine();
        }
    }
}
```

OPERATORER BINÆRE FLOW KONTROL

Logiske boolean operatorer (|, ||, &, && og ^)

OR, AND og eklsusivt OR

Udgaverne | og & bruges sjældent.

OPERATORER BINÆRE FLOW KONTROL

OR(||)

Uanset hvilken side af tegnet der er sandt giver udtrykket et sandt svar. Ved | tjekkes om de efterfølgende er korrekte, ved || stopper tjekket efter den første korrekte.

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace ConsoleApplication6
8   {
        0 references
9       class Program
10      {
            0 references
11          static void Main(string[] args)
12          {
13              int hourOfTheDay = 25;
14              if ((hourOfTheDay > 23) || (hourOfTheDay < 0))
15                  Console.WriteLine("The time you entered is invalid.");
16              else
17                  Console.WriteLine("It is getting a bit late, isn't it?");
18              System.Console.ReadLine();
19          }
20      }
21  }
```

OPERATORER BINÆRE FLOW KONTROL

AND(&&)

Boolean AND operatoren && er kun sand hvis *begge* sider er korrekte.

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace ConsoleApplication6
8   {
        0 references
9       class Program
10      {
            0 references
11          static void Main(string[] args)
12          {
13              int hourOfTheDay = 10;
14              if ((7 < hourOfTheDay) && (hourOfTheDay < 24))
15                  System.Console.WriteLine("Hi-Ho, Hi-Ho, it's off to work we go.");
16              else
17                  Console.WriteLine("It is getting a bit late, isn't it?");
18              System.Console.ReadLine();
19          }
20      }
21  }
```

OPERATORER BINÆRE FLOW KONTROL

Exclusive OR (^)

Kinserhatten ^, er "exclusive OR" (XOR) operatoren. Når den bruges på to Boolean statements bliver den kun sand når kun den ene af dem er det.

Boolean XOR tjekker *altid* begge sider.

Left Operand	Right Operand	Result
True	True	False
True	False	True
False	True	True
False	False	False

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace ConsoleApplication6
8   {
        0 references
9       class Program
10      {
            0 references
11          static void Main(string[] args)
12          {
13              Console.WriteLine(true ^ false);
14              Console.WriteLine(false ^ false);
15              System.Console.ReadLine();
16          }
17      }
18  }
```

OPERATORER
UNÆRE
FLOW KONTROL

Logical Negation Operator (!)

Logical negation operator, eller **NOT** operator, !, vender en bool værdi om.

Den er unær så den kræver kun én operant.

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace ConsoleApplication6
8   {
        0 references
9       class Program
10      {
            0 references
11          static void Main(string[] args)
12          {
13              bool valid = false;
14              bool result = !valid;
15              System.Console.WriteLine($"Result = { result }");
16              System.Console.ReadLine();
17          }
18      }
19  }
```

OPERATORER TERNÆRE FLOW KONTROL

Null-Coalescing Operator (??)

Null-coalescing operatoren bruges når hvis den ene værdi er null så en anden skal bruges.

expression1 ?? expression2

Denne operator kører også en slags kortslutning.
Hvis expression1 ikke er null er resultatet af operationen dets værdi og det andet udtryk behandles ikke. Hvis expression1 *er* null bliver expression2 værdien af operatoren.

OPERATORER TERNÆRE FLOW KONTROL

Null-Conditional Operator (?.)

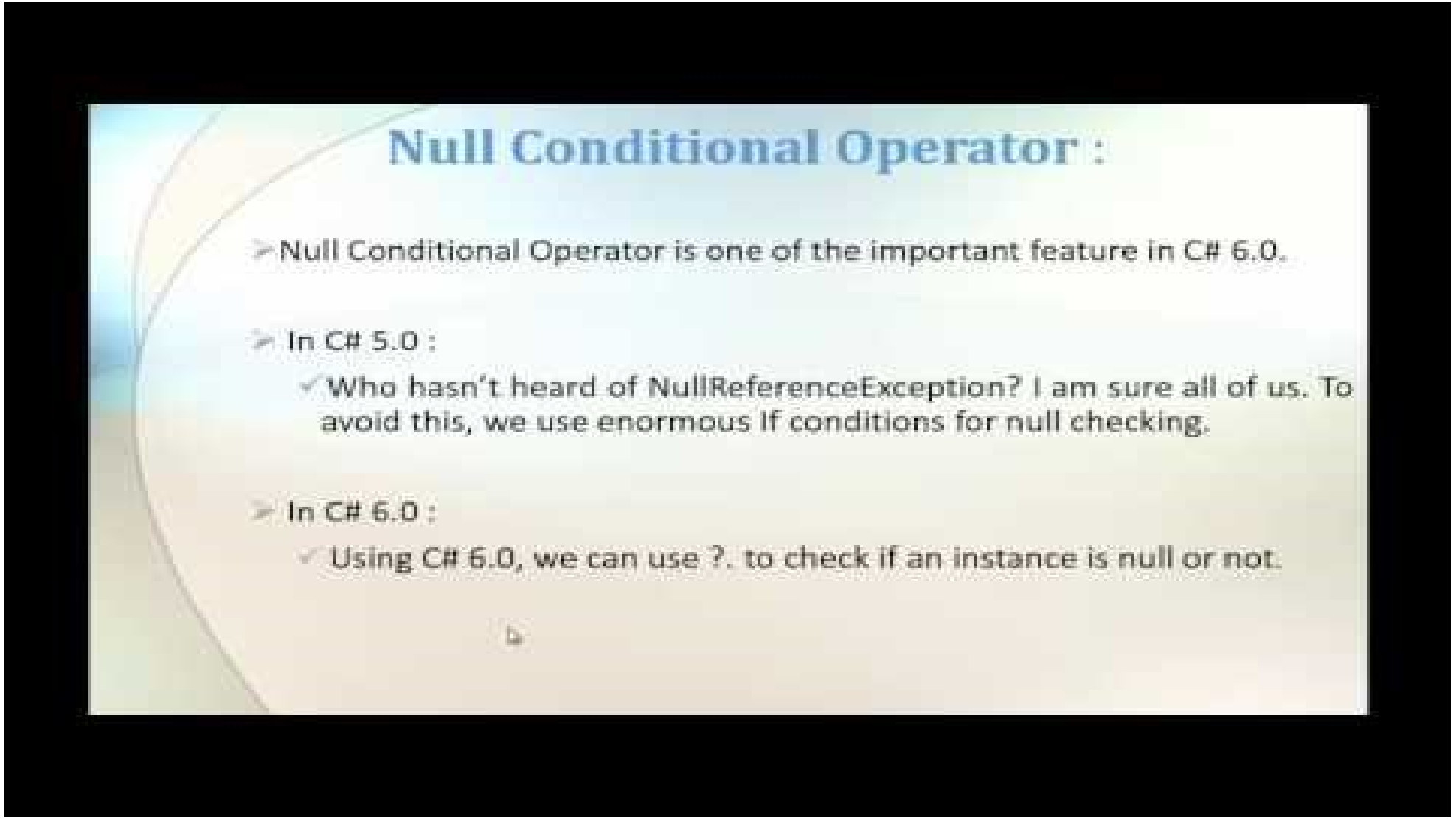
Når man benytter en metode på en værdi, der er null, vil runtime smide en System.NullReferenceException, som næsten altid viser en fejl i programmeringens logik.

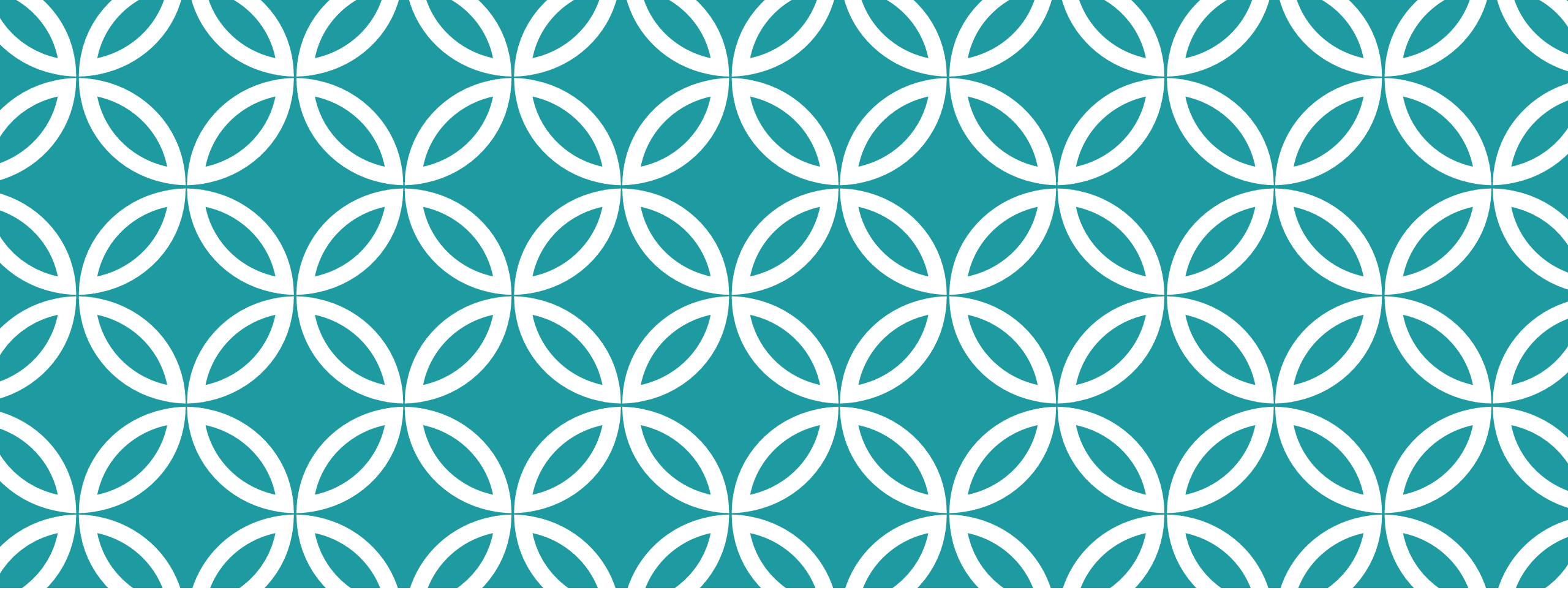
For at anerkende dette problem har C# **null-conditional operator**. Den kontrollerer, om operand (de første i eksemplet herunder) er null før den kalder metoden eller property (length i eksemplet).
Den logisk ækvivalente eksplicitte kode ville være følgende (selvom værdien af args i C # 6.0 syntaks kun evalueres én gang)

```
(args != null) ? (int?)args.Length : null
```

Hvad gør null-conditional operator praktisk er at den kan lænkes.
Hvis man kalder args [0] ?. ToLower ().StartsWith ("File:") vil både ToLower () og StartsWith() blive kaldt hvis args [0] ikke er nul.
Når udtrykket er lænket vil det betyde at hvis den første operand er nul vil udtrykkets evaluering kortsluttes, og ingen yderligere kald i udtrykket vil forekomme.

OPERATORER TERNÆRE FLOW KONTROL





LØKKER

Løkker får verden til at køre rundt

LØKKER

Løkker

En løkke er når en blok kode køres flere gange.

Termen loop body henviser til et statement, typisk en kode blok, der køres i et loop indtil afslutnings-kravet er mødt.

Forskellige former for løkker

Man bruger while til at gentage (iterate) så længe kravet er true.

En for løkke bruges mest hensigtsmæssigt, når antallet af gentagelser er kendt, såsom når der tælles fra 0 til n.

En do/while ligner en while-løkke, men den vil altid udføre løkken kroppen mindst én gang.

LØKKER WHILE

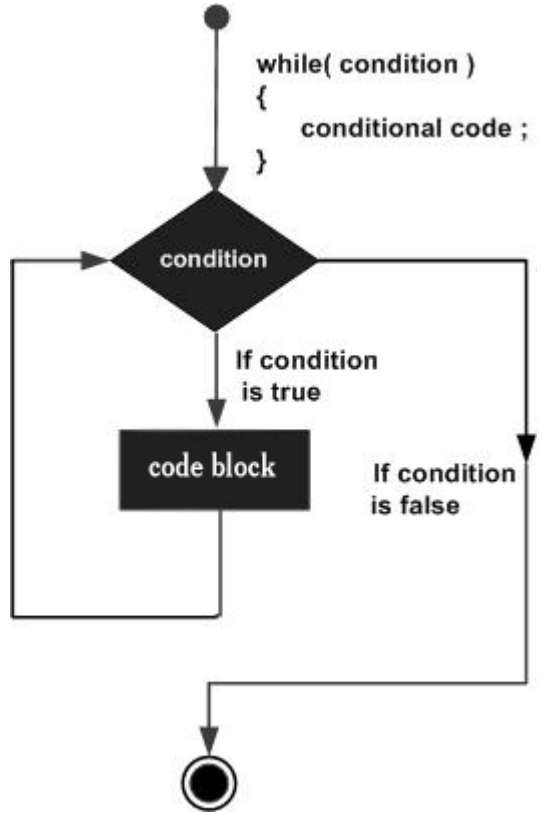
While løkker

While løkken er det enkleste betingede loop. Den almindelige form af while-sætningen er:

while (*condition*)

statement

Loopet kører det statement, der danner kroppen i udtrykket, rundt så længe at conditionen (der *skal* være boolean) er true.
Hvis den bliver *false* dropper udførelsen af koden kroppen og går videre til koden efter loop statementet.



LØKKER WHILE

While løkker

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace ConsoleApplication7
8   {
        0 references
9       class Program
10      {
            0 references
11          static void Main(string[] args)
12          {
13              /* local variable definition */
14              int a = 10;
15
16              /* while loop execution */
17              while (a < 20)
18              {
19                  Console.WriteLine("value of a: {0}", a);
20                  a++;
21              }
22              Console.ReadLine();
23          }
24      }
25  }
```

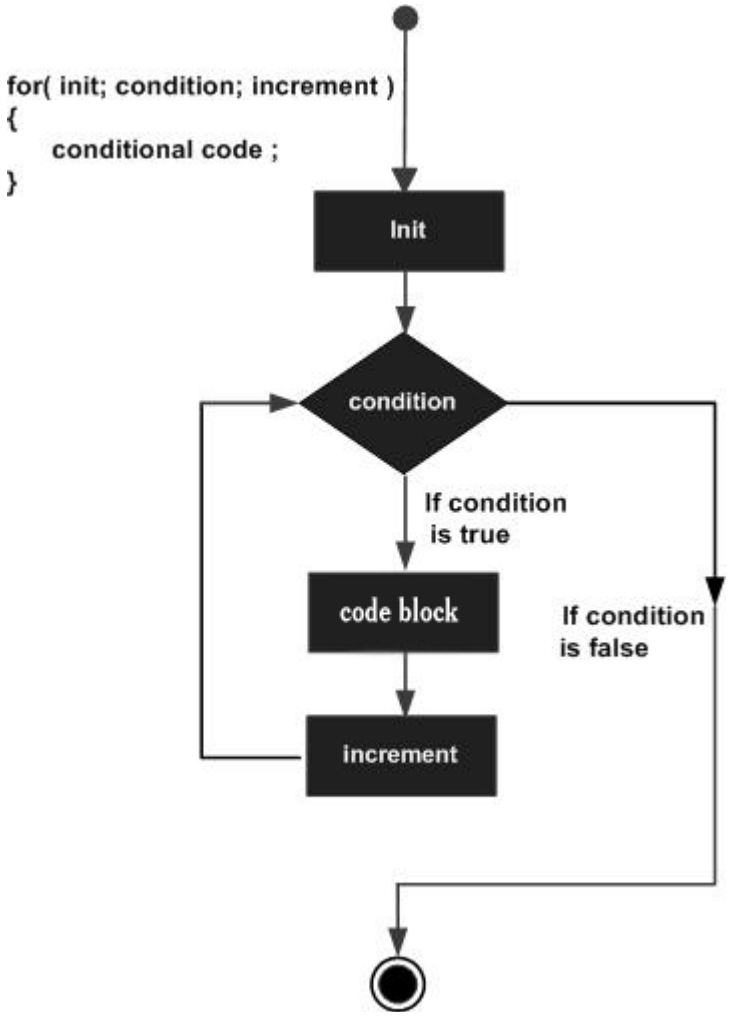
LØKKER FOR

For løkker

En for-løkke gentager en kode blok indtil en bestemt betingelse er nået.

I forhold til while løkken har for-løkken indbygget syntaks for initialisering, forøgelse og afprøvning af værdien af en tæller, kaldet loop variablen.

Fordi der er en specifik placering i loopets syntaks for en tilvækst operation, anvendes increment og decrement operatører ofte som en del af en for-løkke.



LØKKER FOR

For løkker

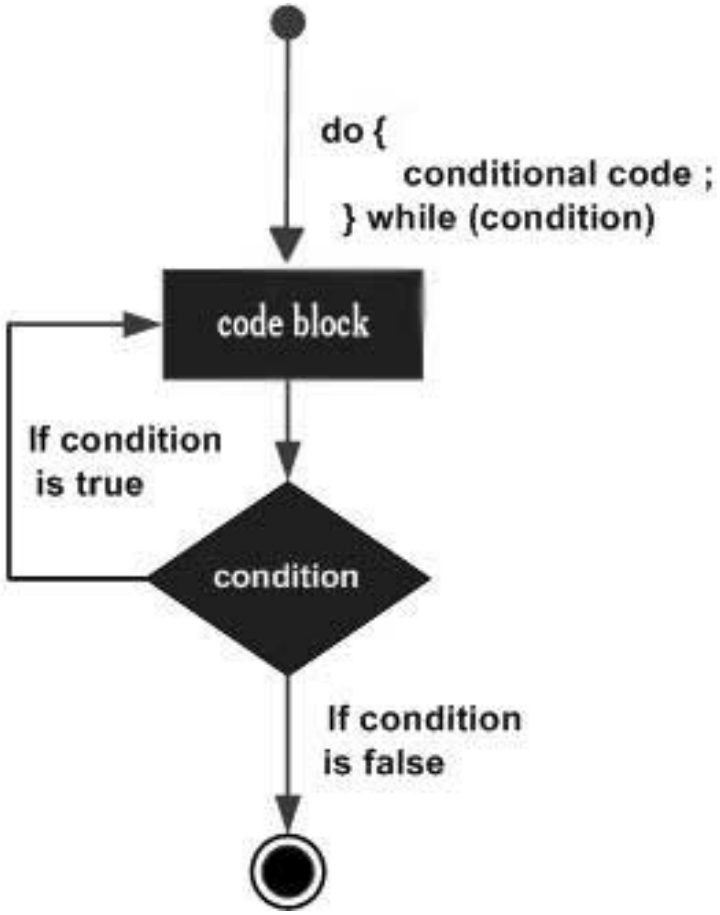
```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace ConsoleApplication7
8   {
        0 references
9       class Program
10      {
            0 references
11          static void Main(string[] args)
12          {
13              /* for loop execution */
14              for (int a = 10; a < 20; a = a + 1)
15              {
16                  Console.WriteLine("value of a: {0}", a);
17              }
18              Console.ReadLine();
19          }
20      }
21  }
```

LØKKER
DO... WHILE

Do... while løkker

Do / while-løkken er meget lig while løkken, bortset fra at do / while løkken foretrækkes, når antallet af gentagelser er fra 1 til n, og n er ikke er kendt når iterationen begynder.

- Dette opstår ofte når man spørger en bruger efter input.



LØKKER
DO… WHILE

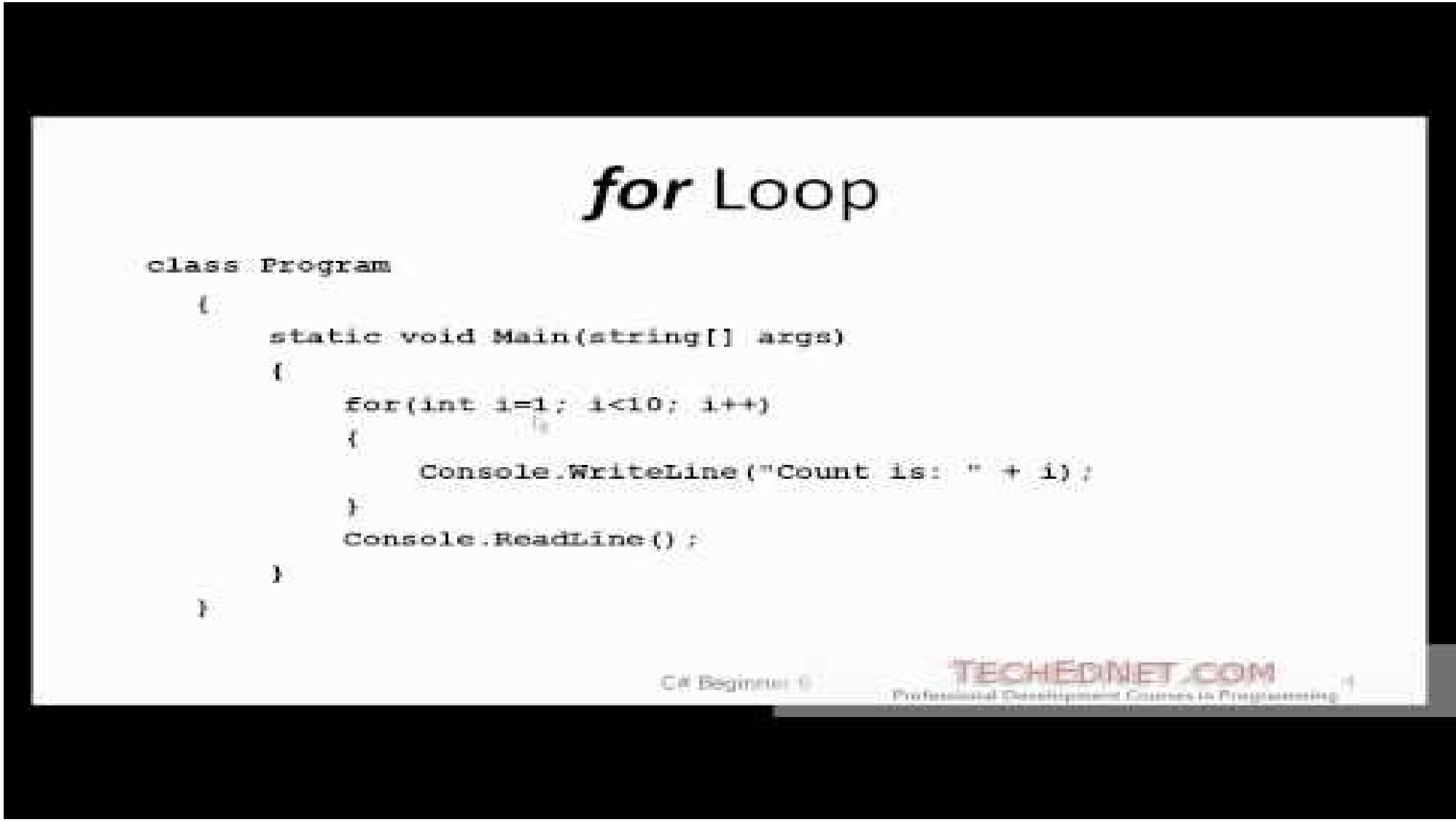
Do… while løkker

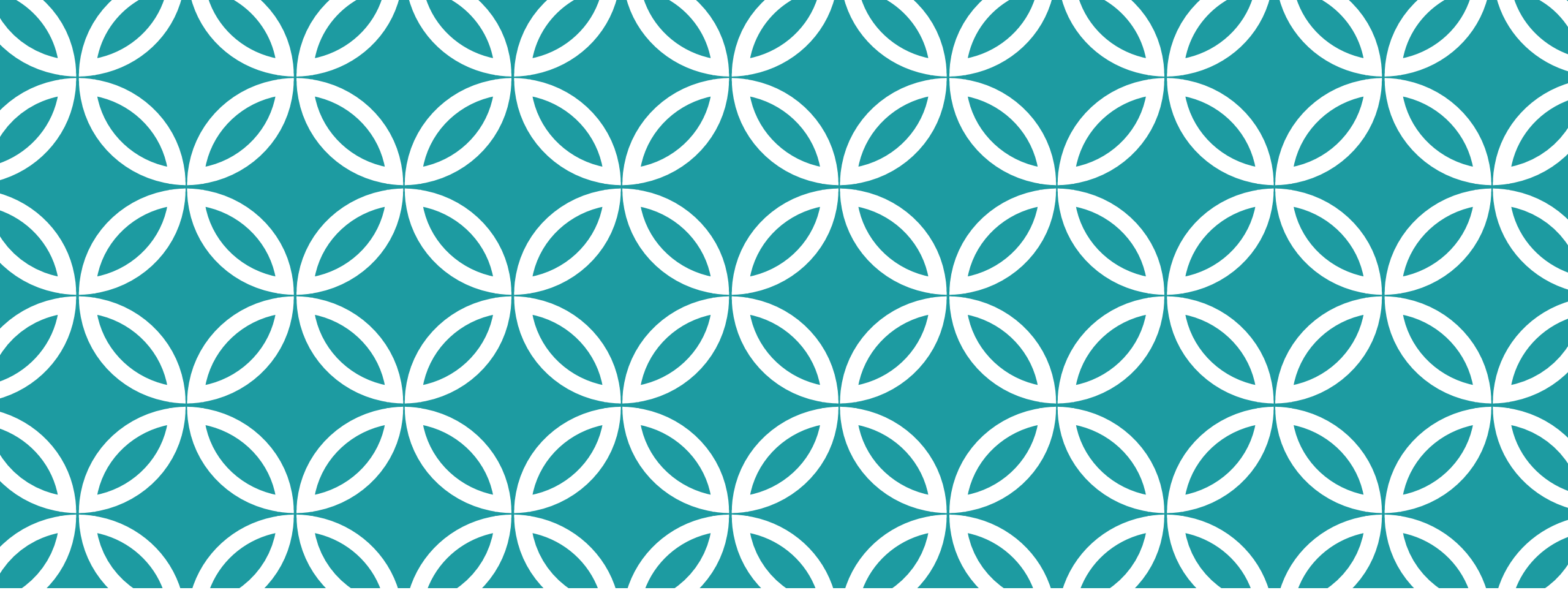
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication7
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            /* local variable definition */
            int a = 10;

            /* do loop execution */
            do
            {
                Console.WriteLine("value of a: {0}", a);
                a = a + 1;
            }
            while (a < 20);
            Console.ReadLine();
        }
    }
}
```

LØKKER


for Loop
class Program
{
static void Main(string[] args)
{
for(int i=1; i<10; i++)
{
Console.WriteLine("Count is: " + i);
}
Console.ReadLine();
}
}
C# Beginner
TECHEDNET.COM
Professional Development Courses in Programming



METODER

Sekvenser af kode om samme emne

METODER

En metode er en måde at samle en sekvens af statements der udfører en bestemt handling eller beregner et bestemt resultat.

- Dette giver større struktur og organisation for de statements, der opbygger et program.

Hvert C # program har mindst en klasse med en metode kaldet Main.

For at benytte en metode skal man:

- Definere metoden.
- Kalde metoden.

Vi har brugt metoder hidtil, primært main() metoden, hvori hele vores programmer lå, men med mere komplekse programmer kan det være praktisk eller nødvendigt at benytte flere metoder.

METODER

En metodes grund struktur er

```
<Access Specifier> <Return Type> <Method Name>(Parameter List)
{
   Method Body
}
```

Access Specifier (scope): Bestemmer synligheden af en variabel eller en metode for andre klasser.

Return type: En metode kan returnere en værdi. Hvis metoden ikke returnerer nogen værdier, så er retur typen ugyldig (void).

Method name: Metode navn er et entydigt id, og det er følsomt overfor store og små bogstaver. Det kan ikke være det samme som en anden identifikator erklæret i klassen.

Parameter list (parametre og argumenter): Lukket inde mellem parenteser finder man de parametre, der anvendes til at sende og modtage data fra metoden. Parameterlisten refererer til typen, rækkefølgen og antallet af parametre i metoden. Parametre er valgfri, det vil sige at en metode ikke behøves have nogen parametre.

Method body: Metodens krop indeholder det sæt af instruktioner, der er nødvendige for at gennemføre den ønskede aktivitet.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication7
{
    2 references
    class NumberManipulator
    {
        1 reference
        public int FindMax(int num1, int num2)
        {
            /* local variable declaration */
            int result;

            if (num1 > num2)
                result = num1;
            else
                result = num2;

            return result;
        }
    }

    0 references
    class Test
    {
        0 references
        static void Main(string[] args)
        {
            /* local variable definition */
            int a = 100;
            int b = 200;
            int ret;
            NumberManipulator n = new NumberManipulator();

            //calling the FindMax method
            ret = n.FindMax(a, b);
            Console.WriteLine("Max value is : {0}", ret);
            Console.ReadLine();
        }
    }
}
```

METODER

Rekursive metode kald

En metode kan kalde sig selv. Dette er kendt som rekursion. Følgende er et eksempel, som beregner fakultetet for et givet tal ved hjælp af en rekursiv funktion:

```
9     class NumberManipulator
10    {
          4 references
11        public int factorial(int num)
12        {
13            /* local variable declaration */
14            int result;
15            if (num == 1)
16            {
17                return 1;
18            }
19            else
20            {
21                result = factorial(num - 1) * num;
22                return result;
23            }
24        }
          0 references
25        static void Main(string[] args)
26        {
27            NumberManipulator n = new NumberManipulator();
28            //calling the factorial method
29            Console.WriteLine("Fakultet for 6 er : {0}", n.factorial(6));
30            Console.WriteLine("Fakultet of 7 er : {0}", n.factorial(7));
31            Console.WriteLine("Fakultet of 8 er : {0}", n.factorial(8));
32            Console.ReadLine();
33        }
34    }
```

METODER

En metode er *ikke* et statement selvom man kunne tro det.

- System.Console .WriteLine ("! Hej {0}", System.Console.ReadLine ()); er et enkelt statement, der indeholder to metodeopkald. Et statement indeholder ofte en eller flere udtryk, og i dette eksempel er to af disse udtryk er metodekald. Derfor danner metodekald dele af udsagn.
- Selvom kodning af flere metodekald i et enkelt statement ofte vil reducerer mængden af kode, betyder det ikke nødvendigvis at det øger læsbarheden, og det giver sjældent en betydelig ydelses fordel. Udviklere bør altid favorisere læsbarheden over kortfattethed.

METODER REFACTORING

Det at flytte en række udsagn ind i en metode i stedet for at lade dem være inline i en større metode er en form for refactoring.

Refactoring reducerer kode dobbeltarbejde, fordi man kan kalde metoden fra flere steder i stedet for at duplikere koden.

Refactoring øger dermed også kodens læsbarhed.

Som del af kodning processen det er en bedste praksis øbende at gennemgå ens kode og se efter muligheder for at refactore.

- Dette indebærer at blokke af kode, der er vanskelige at forstå og få et overblik over, flyttes ind i en metode med et navn, der klart definerer kodens adfærd.
- Denne praksis er ofte foretrukket frem for at kommentere en blok af kode, fordi metodens navn tjener til at beskrive, hvad implementeringen gør.

METODER NAMESPACES

Som I har bemærket begynder vi alle C# programmer med at definere et namespace.

Namespaces er en kategoriserings mekanisme til gruppering af alle typer relateret til et bestemt område af funktionalitet.

Namespaces er hierarkiske og kan have vilkårligt mange niveauer i hierarkiet, omend namespaces med mere end en halv snes niveauer er sjældne.

Typisk begynder hierarkiet med et firmanavn, og derefter et produktnavn, og derefter den funktionelle område.

- For eksempel i Microsoft.Win32.Networking, er det yderste navneområde Microsoft, som indeholder et indre navneområde Win32, som igen indeholder endnu et mere dybt indlejret Networking namespace.
- Det burde virke bekendt i forhold til using i starten af vores programmer.

METODER NAMESPACES

Namespaces bruges primært til at organisere typer efter område af funktionalitet så de lettere kan findes og forstås.

De kan dog også anvendes til at undgå Typenavn kollisioner. Compileren kan f.eks. skelne mellem to typer med navnet Button så længe hver type er under forskellige namespaces.

- Således er System.Web.UI.WebControls.Button og System.Windows.Controls.Button forskellige.

Det er ikke altid nødvendigt at angive et namespace når du kalder en metode. Compileren kan f.eks. udlede at hvis det kaldte udtryk optræder i det samme namespace som den kaldte metode er
namespace det samme som det, der indeholder den type.

God programmerings skik

Benyt PascalCasing (hvert ord begynder med stort) til namespace navne.

Overvej at organisere bibliotekets hierarki for kildekode filer så det matcher namespace hierarkiet.

METODER USING

Fulde namespace navne bliver hurtigt lange og klodsede, så man kan ”importere” indholdet af et eller flere namespaces ind i en fil så man ikke behøves angive de fulde navne.

```
using System;
…
Console.WriteLine("Hello, my name is Inigo Montoya");
```

Eksemplet lader os benytte .Console fra System uden at skulle angive det forrest.

Du kan dog ikke benytte elementer fra child namespaces, så hvis man vil have fat i StringBuilder fra System.Text namespacet skal det også kobles på med using System.Text; direktivet eller man må angive typen som System.Text.StringBuilder – ikke bare Text.Stringbuilder!

METODER USING

static direktivet giver mulighed for at udelade både namespace og type navnet fra et medlem af den angivne type.

```
7   namespace ConsoleApplication7
8   {
9       using static System.Console;
            0 references
10      class HeyYou
11      {
                0 references
12          static void Main()
13          {
14              string firstName;
15              string lastName;
16              WriteLine("Hey you!");
17              Write("Enter your first name: ");
18              firstName = ReadLine();
19              Write("Enter your last name: ");
20              lastName = ReadLine();
21              WriteLine($"Your full name is { firstName } { lastName }.");
22              ReadLine();
23          }
24      }
25  }
```

METODER USING

using direktivet giver også mulighed for aliasing af et namespace eller type. Et alias er et alternativt navn, som man kan bruge i teksten som direktivet påvirker.

De to mest almindelige årsager til aliasing er til at kende forskel på to typer, der har det samme navn og til at forkorte et langt
navn.

```
using System;

// Using alias directive for a class.
using AliasToMyClass = NameSpace1.MyClass;

// Using alias directive for a generic class.
using UsingAlias = NameSpace2.MyClass<int>;

namespace NameSpace1
{
    3 references
    public class MyClass
    {
        1 reference
        public override string ToString()
        {
            return "You are in NameSpace1.MyClass.";
        }
    }
}

namespace NameSpace2
{
    3 references
    class MyClass<T>
    {
        1 reference
        public override string ToString()
        {
            return "You are in NameSpace2.MyClass.";
        }
    }
}

namespace NameSpace3
{
    // Using directive:
    using NameSpace1;
    // Using directive:
    using NameSpace2;

    0 references
    class MainClass
    {
        0 references
        static void Main()
        {
            AliasToMyClass instance1 = new AliasToMyClass();
            Console.WriteLine(instance1);

            UsingAlias instance2 = new UsingAlias();
            Console.WriteLine(instance2);

            Console.ReadLine();
        }
    }
}
```

METODER MAIN

Main metode er indgangen til en C # konsol applikation eller et Windows-program. (Biblioteker og tjenester kræver ikke en Main metode som indgang.). Når programmet startes er Main metoden den første metode, der invokes.

Main metode er indgangen i et .exe program; det er hvor program styring starter og slutter.

Main erklæres inde i en klasse eller struct. Main skal være static, og den bør ikke være offentlig. Den omsluttende klasse eller struct er ikke forpligtet til at være statisk.

Main kan enten have en void eller int return type.

Main metoden kan erklæres med eller uden en string [] parameter, der indeholder kommandolinjeargumenter. Når du bruger Visual Studio til at oprette Windows Forms applikationer, kan du tilføje parameteren manuelt eller andre bruge Environment klassen til at få fat i kommandolinjeargumenter. Parametre læses som nul-indekserede kommandolinjeargumenter.
I modsætning til C og C ++, er navnet på programmet ikke behandlet som det første kommando-line argument.

METODER MAIN

Main metode er indgangen til en C # konsol applikation eller et Windows-program. (Biblioteker og tjenester kræver ikke en Main metode som indgang.). Når programmet startes er Main metoden den første metode, der invokes.

Main metode er indgangen i et .exe program; det er hvor program styring starter og slutter.

Main erklæres inde i en klasse eller struct. Main skal være static, og den bør ikke være offentlig. Den omsluttende klasse eller struct er ikke forpligtet til at være statisk.

Main kan enten have en void eller int return type.

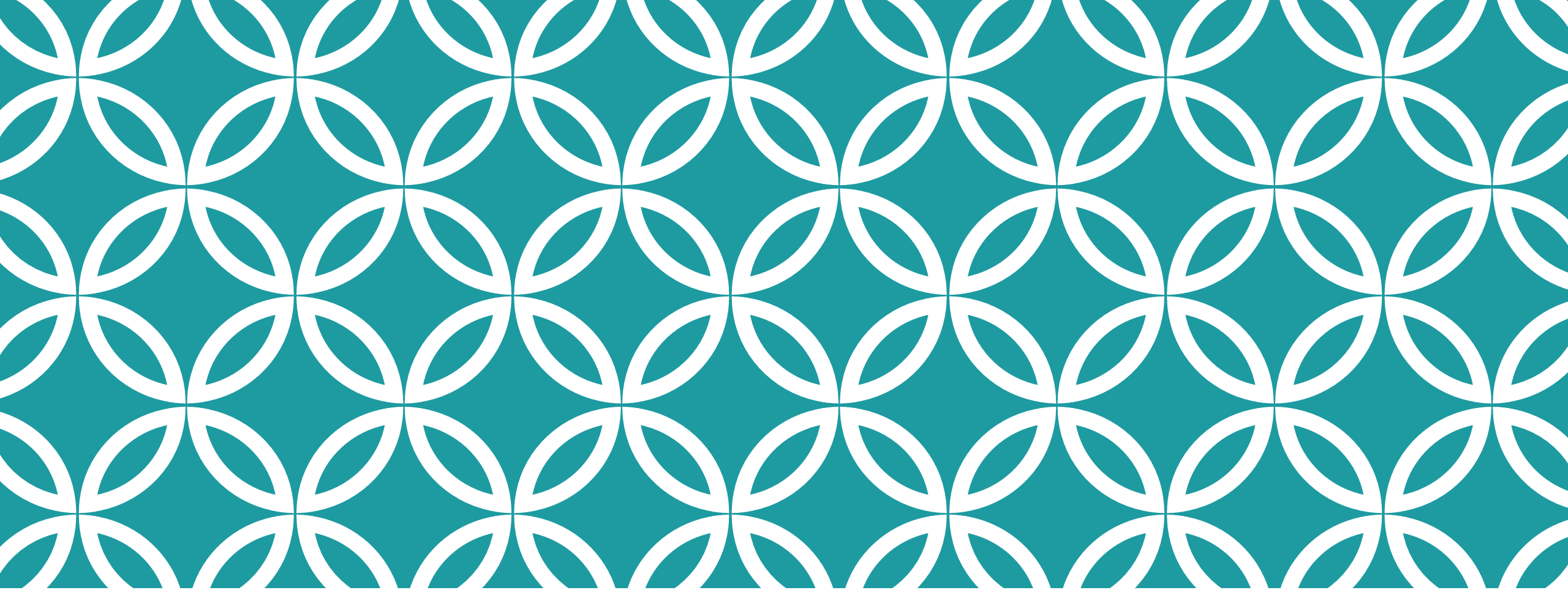
Main metoden kan erklæres med eller uden en string [] parameter, der indeholder kommandolinjeargumenter. Når du bruger Visual Studio til at oprette Windows Forms applikationer, kan du tilføje parameteren manuelt eller andre bruge Environment klassen til at få fat i kommandolinjeargumenter. Parametre læses som nul-indekserede kommandolinjeargumenter.
I modsætning til C og C ++, er navnet på programmet ikke behandlet som det første kommando-line argument.

METODER MAIN

```
using System;

0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        if (args == null)
        {
            Console.WriteLine("args is null"); // Check for null array
        }
        else
        {
            Console.Write("args length is ");
            Console.WriteLine(args.Length); // Write array length
            for (int i = 0; i < args.Length; i++) // Loop through array
            {
                string argument = args[i];
                Console.Write("args index ");
                Console.Write(i); // Write index
                Console.Write(" is [");
                Console.Write(argument); // Write string
                Console.WriteLine("]");
            }
        }
        Console.ReadLine();
    }
}
```

METODER



C'S HISTORIE

Afbræk

C'S HISTORIE

LEKTIE

Kig på dette til næste gang

LEKTIE

Se og løs opgaverne til https://mva.microsoft.com/en-US/training-courses/programming-in-c-jump-start-14254 - 02

Læs:

- https://www.tutorialspoint.com/csharp/csharp_operators.htm,
- https://www.tutorialspoint.com/csharp/csharp_decision_making.htm,
- https://www.tutorialspoint.com/csharp/csharp_methods.htm og
- https://www.tutorialspoint.com/csharp/csharp_classes.htm

KILDER

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

KILDER

Operatorer

http://www.functionx.com/csharp2/conditions/Lesson01d.htm

https://msdn.microsoft.com/en-us/library/hh147286(v=vs.88).aspx

https://msdn.microsoft.com/en-us/library/6a71f45d(VS.71).aspx

https://msdn.microsoft.com/da-dk/library/ty67wk28.aspx

https://msdn.microsoft.com/en-us/library/zakwfxx4(v=vs.100).aspx

https://msdn.microsoft.com/da-dk/library/ms173224(v=vs.100).aspx

KILDER

Løkker

https://www.tutorialspoint.com/csharp/csharp_while_loop.htm

https://www.tutorialspoint.com/csharp/csharp_for_loop.htm

https://www.tutorialspoint.com/csharp/csharp_do_while_loop.htm

https://youtu.be/Yvc_iTy_BtI

Metoder

https://www.tutorialspoint.com/csharp/csharp_methods.htm

https://msdn.microsoft.com/da-dk/library/sf0df423.aspx

KILDER

Metoder

https://www.tutorialspoint.com/csharp/csharp_methods.htm

https://msdn.microsoft.com/da-dk/library/sf0df423.aspx

https://msdn.microsoft.com/da-dk/library/c3ay4x3d.aspx

https://www.dotnetperls.com/main

https://msdn.microsoft.com/da-dk/library/acy3edy3.aspx

https://youtu.be/QwygwfqOHsI

C's historie

https://youtu.be/azDM109_j-g