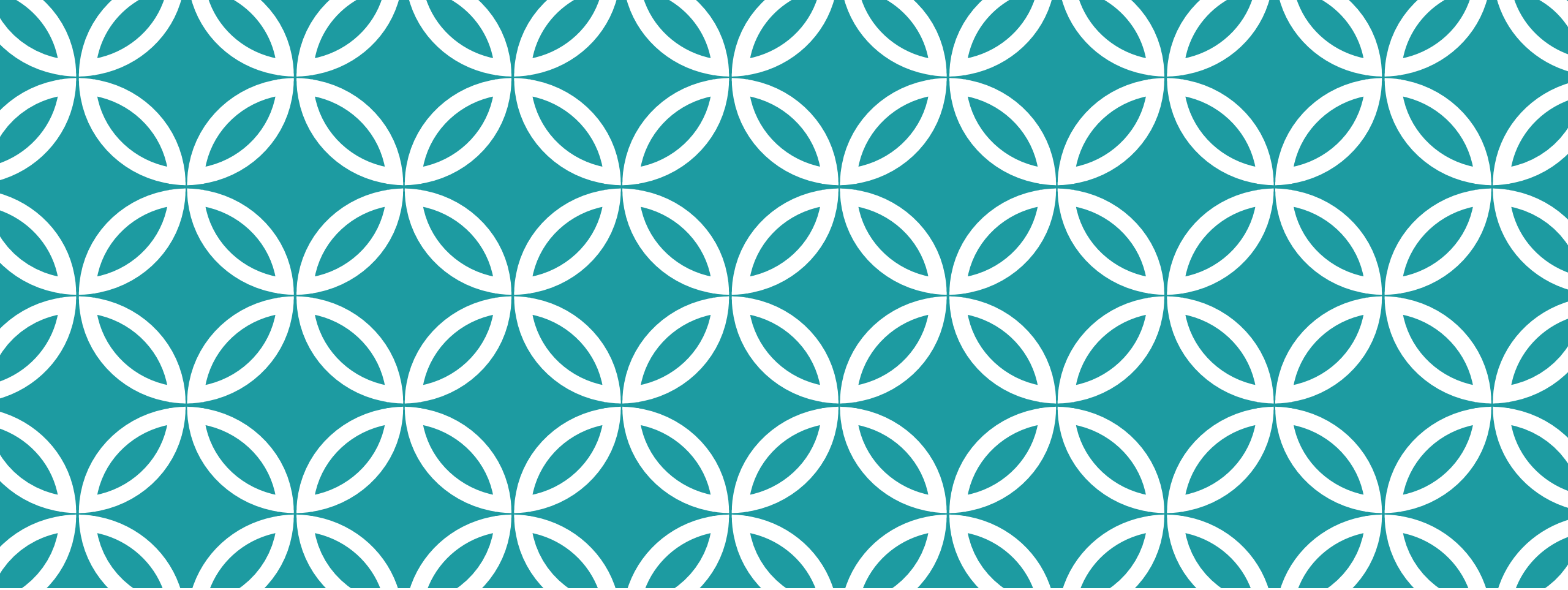
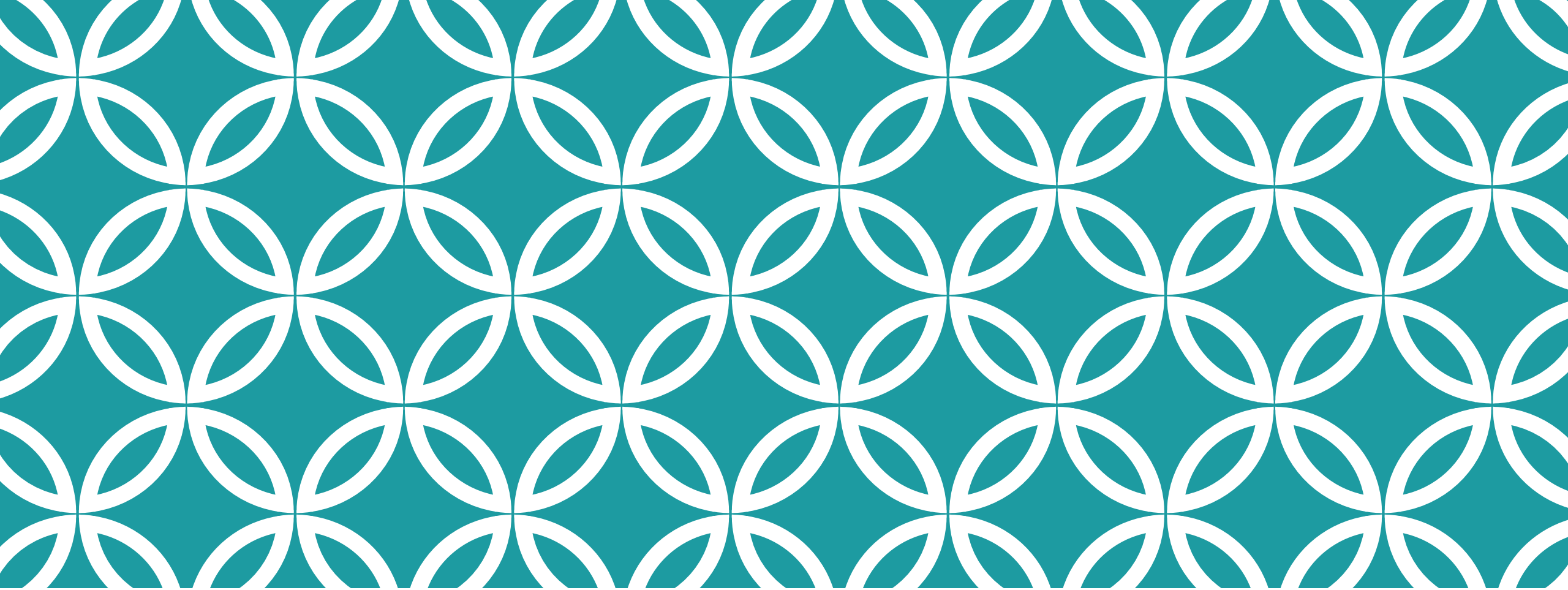
DATA TYPER
OPERATORER
PRAKSIS EKSEMPEL UNITY

Grundlæggende programmering
Lektion 3



LEKTIER

UML struktur diagrammer om biler



DATA TYPER

Forskellige former for data opbevares forskelligt

DATA TYPER

Variablerne i C# er inddelt i tre typer

Værdi typer / Value types

Reference typer / Reference types

Markør typer / Pointer types

DATA TYPER
VÆRDI TYPER

Value types variabler kan direkte få en værdi. De er afledt af klassen class **System.ValueType**.

Det kan være data som i**nt, char** og **float,** der gemmer tal, bogstaver og komma tal respektivt.

Når man deklarerer en **int** type allokerer systemet automatisk hukommelse til at gemme værdien

DATA TYPER
VÆRDI TYPER

Type	Represents	Range	Default Value
bool	Boolean value	True or False	False
byte	8-bit unsigned integer	0 to 255	0
char	16-bit Unicode character	U +0000 to U +ffff	'\0'
decimal	128-bit precise decimal values with 28-29 significant digits	$(-7.9 \times 10^{28}$ to $7.9 \times 10^{28}) / 10^{0}$ to 28	0.0M
double	64-bit double-precision floating point type	$(+/-)5.0 \times 10^{-324}$ to $(+/-)1.7 \times 10^{308}$	0.0D
float	32-bit single-precision floating point type	-3.4×10^{38} to $+ 3.4 \times 10^{38}$	0.0F
int	32-bit signed integer type	-2,147,483,648 to 2,147,483,647	0
long	64-bit signed integer type	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0L
sbyte	8-bit signed integer type	-128 to 127	0
short	16-bit signed integer type	-32,768 to 32,767	0
uint	32-bit unsigned integer type	0 to 4,294,967,295	0
ulong	64-bit unsigned integer type	0 to 18,446,744,073,709,551,615	0
ushort	16-bit unsigned integer type	0 to 65,535	0

DATA TYPER
VÆRDI TYPER

For at få den eksakte størrelse på en type eller en variabel på en bestemt platform kan man benytte **sizeof** metoden. Udtrykket expression *sizeof(type)* giver opbevarings størrelse på objektet eller typen i bytes. Husk at størrelsen på typen er fast uanset indhold.

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   using System;
8   using System.Collections.Generic;
9   using System.Linq;
10  using System.Text;
11  using System.Threading.Tasks;
12
13  namespace DataTypeApplication
14  {
        0 references
15      class Program
16      {
            0 references
17          static void Main(string[] args)
18          {
19              int i = 123;
20              Console.WriteLine("The size of short is {0}.", sizeof(short));
21              Console.WriteLine("The size of int is {0}.", sizeof(int));
22              Console.WriteLine("The size of long is {0}.", sizeof(long));
23              Console.ReadLine();
24          }
25      }
26  }
```

DATA TYPER
REFERENCE TYPER

En reference type indeholder ikke faktisk data gemt i en variabel men indeholder en reference til variablerne.

De refererer altså til en plads i hukommelsen. Således kan flere variabler af reference typerne referere til sammes plads i hukommelsen. Hvis dataen på denne plads i hukommelsen ændres af en af variablerne vil de andre automatisk ændre indhold.
Et eksempel på **built-in** reference typer er: **object, dynamic** og **string**.

DATA TYPER
OBJEKT TYPER

Object Type er den grundlæggende base klasse for alle datatyper i C#
Object er et alias for System.Object klassen. Objekt typer kan få værdier fra andre typer, value typer, reference typer, predefined eller user-defined typer. Men før den kan tildeles disse værdier, har den dog brug for type konvertering.

Når en value type konverteres til en object type kaldes det **boxing** og den anden vej rundt, når en object type konverteres til en value type, kaldes det **unboxing**

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

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

namespace DataTypeApplication
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            object obj;
            obj = 100; // this is boxing
            Console.WriteLine("The stored number is {0}",obj);
            Console.ReadLine();
        }
    }
}
```

DATA TYPER

DYNAMISKE TYPER

Du kan gemme hvilken som helst værdi i dynamic data type variabler. Type checking af disse typer variabler tager tid ved run-time.

Dynamiske typer ligner objekt typer bortset fra at type checking for object type variabler finder sted ved compilering, hvor dynamic type variabler finder sted ved run time.

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   using System;
8   using System.Collections.Generic;
9   using System.Linq;
10  using System.Text;
11  using System.Threading.Tasks;
12
13  namespace DataTypeApplication
14  {
        0 references
15      class Program
16      {
            0 references
17          static void Main(string[] args)
18          {
19              dynamic d = 20;
20              Console.WriteLine("The stored number is {0}",d);
21              Console.ReadLine();
22          }
23      }
24  }
```

DATA TYPER
STRENG TYPER

String Type lader en angive en hvlken som helst string værdi til en variabel. String type er et alias for System.String klassen. Den er afledt af en objekt type. Værdien i en string kan angives ved at benytte string literals i to former: quoted og @quoted.

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   using System;
8   using System.Collections.Generic;
9   using System.Linq;
10  using System.Text;
11  using System.Threading.Tasks;
12
13  namespace DataTypeApplication
14  {
        0 references
15      class Program
16      {
            0 references
17          static void Main(string[] args)
18          {
19              String str = "This is a\nmulti-line\nstring";
20              Console.WriteLine(str);
21              Console.ReadLine();
22          }
23      }
24  }
```

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   using System;
8   using System.Collections.Generic;
9   using System.Linq;
10  using System.Text;
11  using System.Threading.Tasks;
12
13  namespace DataTypeApplication
14  {
        0 references
15      class Program
16      {
            0 references
17          static void Main(string[] args)
18          {
19              string str = @"This is NOT a\nmulti-line\nstring";
20              Console.WriteLine(str);
21              Console.ReadLine();
22          }
23      }
24  }
```

DATA TYPER
STRENG TYPER

Strenge understøtter indlejrede (embedded) udtryk når man benytter string interpolation format.

Hvis du vil finde længden på en streng bruger du **length,** der er en read only egenskab der hverken kan sættes eller kræver nogen parametre.

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace ConsoleApplication5
8   {
        0 references
9       class Program
10      {
            0 references
11          static void Main(string[] args)
12          {
13              string word;
14              System.Console.Write("Enter a word: ");
15              word = System.Console.ReadLine();
16              System.Console.WriteLine(
17                  $"The word \"{word}\" is"
18                  + $" {word.Length} characters.");
19              Console.ReadLine();
20          }
21      }
22  }
```

DATA TYPER
MARKØR TYPER

Markør/pointer type variabler gemmer hukommelses adressen for en anden type. Pointers i C# har de samme muligheder som pointers i C eller C++.

```
int* myVariable;
```

Udtrykket *myVariable henviser til int variablen der findes på adressen indeholdt i myVariable.

Pointers kan dog let lede til usikker kode, hvilket vi kommer ind på senere.

DATA TYPER

OPGAVE

Lav et program der indebeholder følgende

- Erklær en double variabel kaldet "penge" og tildel den værdien 1075.50
- Erklær en string variabel og tildel den værdien: "Jeg har"
- Erklær en ny string variabel, og tildel den værdien: " kr. i banken"
- Udskriv følgende: "Det er ikke for at prale men…" og en ny linje med "Jeg har 1075.50 kr. i banken"
  - Du skal her bruge de 3 variabler du har lavet, og udskrive dem på samme linie
  - Husk at der er forskel på Console.Write og Console.WriteLine

DATA TYPER

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace ConsoleApplication5
8   {
        0 references
9       class Program
10      {
            0 references
11          static void Main(string[] args)
12          {
13              double penge = 1075.50;
14              string jh  = "Jeg har ";
15              string kr = "kr i banken.";
16              Console.WriteLine("Det er ikke for at prale men...");
17              Console.Write(jh);
18              Console.Write(penge);
19              Console.Write(kr);
20              Console.ReadLine();
21          }
22      }
23  }
```

DATA TYPER
KONVERTERING

Vi så tidligere hvordan man kunne unboxe en objekt type så den blev en data type. Man kan generelt konvertere mellem data typer, men husk kun at gøre det når det giver mening da man let får data tab!

Tag eksemplet **long** til **int**, der går vi fra værdier så store som 9.223.372.036.854.775.808 til et maks på 2.147.483.647.

En konversion der resulterer i et sådant tab kræver et **explicit cast** hvor en der ikke resulterer i data-tab og dermed ingen exception er en **implicit conversion**.

DATA TYPER
KONVERTERING EKSPLICIT

Explicit cast benytter **cast** operatoren, hvor man ved at angive det man vil konvertere i parentesen viser systemet at man virkelig vil det.

```
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              double x = 1234.7;
14              int a;
15              a = (int)x;
16              Console.WriteLine(a);
17              Console.ReadLine();
18          }
19      }
20  }
```

DATA TYPER
KONVERTERING IMPLICIT

Ved en implicit konvertering kan compilere ikke se nogen fejl i konverteringen, så den udfører den. **int** til **long** f.eks.

Der er dog ekstra muligheder som system.convert, der dog kunne virker mellem få typer, og ToString() der virker med alt.

```
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 boolean = true;
14              string text = boolean.ToString();
15              System.Console.WriteLine(text);
16              Console.ReadLine();
17          }
18      }
19  }
```

DATA TYPER KONVERTERING

OPGAVE

Lav et program der indebeholder følgende

- Bed brugeren om tre bogstaver og vis dem i omvendt rækkefølge.
- Husk kommandoen Convert.ToChar og kombiner den med Console.ReadLine

DATA TYPER KONVERTERING

```
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              char letter, letter2, letter3;
14
15              Console.Write("Enter letter: ");
16              letter = Convert.ToChar(Console.ReadLine());
17
18              Console.Write("Enter letter: ");
19              letter2 = Convert.ToChar(Console.ReadLine());
20
21              Console.Write("Enter letter: ");
22              letter3 = Convert.ToChar(Console.ReadLine());
23
24              Console.WriteLine("{0} {1} {2}", letter3, letter2, letter);
25
26              Console.ReadLine();
27          }
28      }
29  }
```

DATA TYPER
ARRAYS

I C # erklærer du arrays med firkantede parenteser

Først angiver man typen af array.

Dette følges af åbne og lukkede firkantede parenteser.

Så angiver man navnet på variablen.

```
string[] sprog;

int[] telefonnummer;

bool [] fredag
```

Arrays kan være i flere dimensioner, C# understøtter single-dimensional arrays, multidimensional arrays (rectangular arrays), og array-of-arrays (jagged arrays).S

DATA TYPER
ARRAYS

Single-dimensional arrays

```
string[] name = new string[]{"Erik"};
```

Multidimensional arrays (rectangular arrays)

```
string[,] names = new string[]{" Erik","Britta"};
```

Array-of-arrays (jagged arrays)

```
byte[][] scores = new byte[5][];
for (int x = 0; x < scores.Length; x++)
{
        scores[x] = new byte[4];
}
```

DATA TYPER
ARRAYS

Single-dimensional arrays

```
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              string[] name = new string[]{"Erik"};
14
15              foreach (string item in name)
16              {
17                  Console.WriteLine(item.ToString());
18              }
19
20              Console.ReadLine();
21          }
22      }
23  }
```

DATA TYPER
ARRAYS

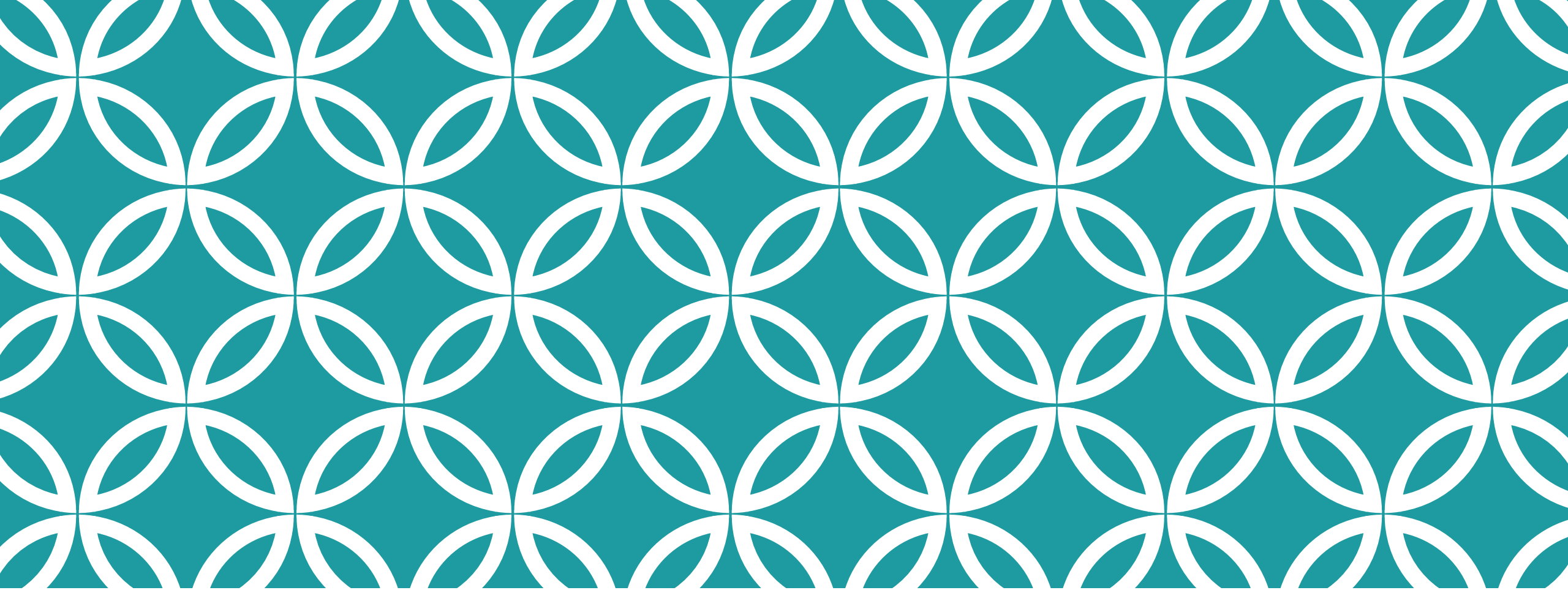
Multidimensional arrays (rectangular arrays)

```
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              string[] name = new string[]{"Erik","Britta"};
14
15              foreach (string item in name)
16              {
17                  Console.WriteLine(item.ToString());
18              }
19
20              Console.ReadLine();
21          }
22      }
23  }
```

DATA TYPER
ARRAYS

Array-of-arrays (jagged arrays)

```
9   class Program
10  {
        0 references
11      static void Main(string[] args)
12      {
13          // Declare local jagged array with 3 rows.
14          int[][] jagged = new int[3][];
15
16          // Create a new array in the jagged array, and assign it.
17          jagged[0] = new int[2];
18          jagged[0][0] = 1;
19          jagged[0][1] = 2;
20
21          // Set second row, initialized to zero.
22          jagged[1] = new int[1];
23
24          // Set third row, using array initializer.
25          jagged[2] = new int[3] { 3, 4, 5 };
26
27          // Print out all elements in the jagged array.
28          for (int i = 0; i < jagged.Length; i++)
29          {
30              int[] innerArray = jagged[i];
31              for (int a = 0; a < innerArray.Length; a++)
32              {
33                  Console.Write(innerArray[a] + " ");
34              }
35              Console.WriteLine();
36          }
37          Console.ReadLine();
38      }
39  }
```



OPERATORER

Syntaks til at udføre forskellige beregninger og handlinger

Booleans og hvorfor de er essentielle og praktiske

OPERATORER

Operatorer bruges til at udføre matematiske eller logiske operationer på værdier (eller variabler) kaldet operander for at producere en ny værdi, kaldet resultatet.

```
int difference = 4 – 2;
```

Her er – operatoren og resultatet 2 gemmes i integer data typen difference.

Operatorer er generelt inddelt i tre kategorier

unære / unary

binære / binary

Ternære / tertiary

svarende til antallet af operander (en, to, og tre henholdsvis).

OPERATORER
UNÆRE

Plus og minus unære operatorer (+, -)

Hvis man har brug for at ændre en værdi fra positiv til negativ (eller omvendt) kan C# gøre det med den unære operator –

Den undære operator + har sjældent en effekt og er mest med for god ordens skyld.

OPERATORER BINÆRE

Aritmetiske binære operatorer (+, -, *, /, %)

Disse bruges til at foretage beregninger med to værdier.

```
8   {
        0 references
9       class Program
10      {
            0 references
11          static void Main(string[] args)
12          {
13              int numerator;
14              int denominator;
15              int quotient;
16              int remainder;
17
18              System.Console.Write("Enter the numerator: ");
19              numerator = int.Parse(System.Console.ReadLine());
20
21              System.Console.Write("Enter the denominator: ");
22              denominator = int.Parse(System.Console.ReadLine());
23
24              quotient = numerator / denominator;
25              remainder = numerator % denominator;
26
27              System.Console.WriteLine(
28              $"{numerator} / {denominator} = {quotient} with remainder {remainder}");
29
30              Console.ReadLine();
31          }
32      }
33  }
```

OPERATORER BINÆRE

Aritmetiske binære operatorer (+, -, *, /, %)

I eksemplet bliver divisionen og resten operationerne gennemført før opgaverne. Den rækkefølge, som de operatorerne udføres ier bestemt af deres forrang (precedence) og associativitet (associavity).

Rangen for operatørerne hidtil anvendt er_

1. *, / og % har højeste prioritet.
2. + og - har lavere prioritet.
3. = har den laveste rang af disse seks operatører.

Man kan også benytte + til at sætte andet end tal sammen, f.eks. flere strenge som vi har gjort tidligere.

OPERATORER BINÆRE

Compound Assignment Operators (+=, -=, *=, /=, %=)
Compound assignment operators kombinerer standard binære operator beregninger med en assignment operator.

Således giver

```
int x = 123;
x = x + 2;
```

og

```
int x = 123;
x += 2;
```

det samme.

Der findes talrige andre ”compund assignment" operatører til at give lignende funktionalitet. Du kan således også bruge assignment operatoren sammen med subtraktion, multiplikation, division, og resten (remainder) operatører.

OPERATORER UNÆRE

Forøg og formindsk operatorer (++, --)

C # omfatter særlige unære operatorer til forøgelse og formindskelse. Således inkrementerer inkrementeringsoperatoren, ++, en variabel hver gang den anvendes.

```
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 Value = 12;
14
15                  Console.WriteLine("Techniques of incrementing a value");
16                  Console.Write("Value = ");
17                  Console.WriteLine(Value);
18
19                  Value = Value + 1;
20
21                  Console.Write("Value = ");
22                  Console.WriteLine(Value);
23
24              Console.ReadLine();
25          }
26      }
27  }
```

OPERATORER UNÆRE

Forøg og formindsk operatorer (++, --)

```
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 Value = 12;
14
15              Console.WriteLine("Techniques of incrementing a value");
16              Console.Write("Value = ");
17              Console.WriteLine(Value);
18
19              Value++;
20
21              Console.Write("Value = ");
22              Console.WriteLine(Value);
23
24              Console.ReadLine();
25          }
26      }
27  }
```

OPERATORER UNÆRE

Forøg og formindsk operatorer (++, --)

```
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 Value = 12;
14
15              Console.WriteLine("Techniques of incrementing a value");
16
17              Value++;
18              Console.Write("Value = ");
19              Console.WriteLine(Value);
20
21              Value++;
22              Console.Write("Value = ");
23              Console.WriteLine(Value);
24
25              Value++;
26              Console.Write("Value = ");
27              Console.WriteLine(Value);
28
29              Console.ReadLine();
30          }
31      }
32  }
```

OPERATORER UNÆRE

Forøg og formindsk operatorer (++, --)

```
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 Value = 12;
14
15              Console.WriteLine("Techniques of incrementing a value");
16
17              Value++;
18              Console.Write("Value = ");
19              Console.WriteLine(Value);
20
21              Value--;
22              Console.Write("Value = ");
23              Console.WriteLine(Value);
24
25              Value--;
26              Console.Write("Value = ");
27              Console.WriteLine(Value);
28
29              Console.ReadLine();
30          }
31      }
32  }
```

OPERATORER UNÆRE

Forøg og formindsk operatorer (++, --)

Det har betydning om operatoren star foran eller bagved.

Når man skriver en ++ foran en værdi bliver værdien af variablen øget før den bliver kaldt. Hvis man derimod skriver ++ efter øges værdien først efter at den er blevet kaldt.

```
7   namespace ConsoleApplication6
8   {
        0 references
9       class Program
10      {
            0 references
11          static void Main(string[] args)
12          {
13              int Value = 12;
14
15              Console.WriteLine("Techniques of incrementing a value");
16
17              Console.Write("Value = ");
18              Console.WriteLine(Value);
19
20              Console.Write("Value = ");
21              Console.WriteLine(Value++);
22
23              Console.Write("Value = ");
24              Console.WriteLine(Value);
25
26              Console.ReadLine();
27          }
28      }
29  }
```

OPERATORER FLOW KONTROL

Flow kontrol

Selv i simple programmer kan det være nødvendigt at styre rækkefølgen af udførelse af programmet, ikke blot fra start til slut

Al C # kode I har set hidtil har haft én ting til fælles. I hvert tilfælde er programmet udført fra den ene linje til den næste fra top til bund i rækkefølge, intet mangler. Hvis alle programmer arbejdede som dette, så ville man være meget begrænset i hvad man kunne gøre.

For at opnå dette benytter man forgrening og looping. Forgrening udfører kode betinget, afhængigt af resultatet af en evaluering, som "only execute this code if the variable myVal is less than 10."
Looping udfører gentagne gange de samme udsagn, enten et bestemt antal gange, eller indtil en test tilstand har været nået.

OPERATORER FLOW KONTROL

Flow kontrol

Selv i simple programmer kan det være nødvendigt at styre rækkefølgen af udførelse af programmet, ikke blot fra start til slut

Al C # kode I har set hidtil har haft én ting til fælles. I hvert tilfælde er programmet udført fra den ene linje til den næste fra top til bund i rækkefølge, intet mangler. Hvis alle programmer arbejdede som dette, så ville man være meget begrænset i hvad man kunne gøre.

For at opnå dette benytter man forgrening og looping.
Forgrening udfører kode betinget, afhængigt af resultatet af en evaluering, som "only execute this code if the variable myVal is less than 10."
Looping udfører gentagne gange de samme udsagn, enten et bestemt antal gange, eller indtil en test tilstand har været nået.

Begge disse teknikker indebærer anvendelse af boolsk logik.

OPERATORER FLOW KONTROL

if statements

Et if statement er et af de mest udbredte i C#. Det evaluerer et **boolsk udtryk** (**Boolean expression**), et udtryk der resulterer i enten sandt eller) der kaldes **condition.**
Hvis tilstanden er sand så udføres **consequence statement.**

Et if statement kan eventuelt have en else clause der indeholder et **alternativt statement** der udføres hvis condition er falsk.

```
if (condition)
consequence-statement
else
alternative-statement
```

OPERATORER FLOW KONTROL

if statements

```
11          static void Main(string[] args)
12          {
13              int input; // Declare a variable to store the input.
14
15              System.Console.Write(
16              "What is the maximum number " +
17              "of turns in tic-tac-toe?" +
18              "(Enter 0 to exit.): ");
19
20              // int.Parse() converts the ReadLine() - return to an int data type.
21              input = int.Parse(System.Console.ReadLine());
22
23              if (input <= 0) // line 16 - Input is less than or equal to 0.
24                  System.Console.WriteLine("Exiting...");
25              else
26              if (input < 9) // line 20 - Input is less than 9.
27                  System.Console.WriteLine(
28                  $"Tic-tac-toe has more than {input}" +
29                  " maximum turns.");
30              else
31              if (input > 9) // line 26 - Input is greater than 9.
32                  System.Console.WriteLine(
33                  $"Tic-tac-toe has fewer than {input}" +
34                  " maximum turns.");
35              else
36                  // Input equals 9.
37                  System.Console.WriteLine( // line 33
38                  "Correct, tic-tac-toe " +
39                  "has a maximum of 9 turns.");
40              Console.ReadLine();
41          }
42      }
43  }
```

OPERATORER FLOW KONTROL

Code blocks

Med tuborg-klammer {} kan man kombinere statements til et enkelt statement, et **block statement** eller **code block** der fungerer som ét statement.

```
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              Random r = new Random();
14              int n = r.Next(-5, 5);
15
16              Console.WriteLine(n);
17
18              if (n > 0)
19              {
20                  Console.WriteLine("The n variable is positive");
21              }
22          Console.ReadKey();
23          }
24      }
25  }
```

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 (?.)

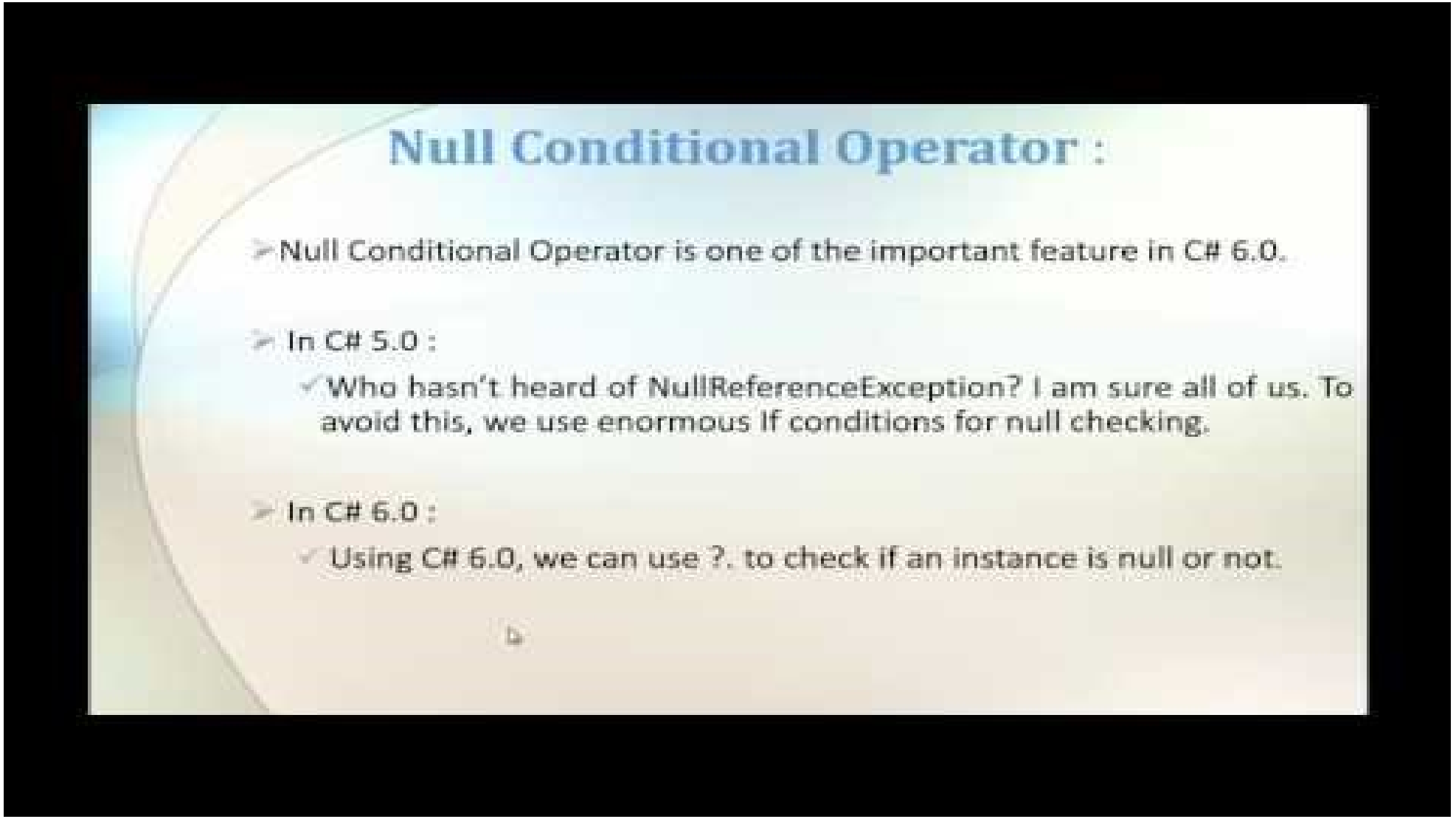
Når man benytter en metode på en værdi, der er null, vil runtime smide en System.NullReferenceException, som næsten altid viser en fejl i programmeringens logik.

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



PRAKSIS EKSEMPEL

Et program der benytter Unity til at lave et 3D spil

PRAKSIS EKSEMPEL

Spil der benytter Unity og er scriptet i C#

Unity miljøet til spil og 3D app udvikling er ekstremt populært, og det benyter C# som scripting sprog – uanset om man vælger at compilere ens app til Windows, Linux eller Mac. Der er også webplayers og mulighed for at benytte Unity til mobil-apps.

PRAKSIS EKSEMPEL

Kort introduktion til Unitys brug af C#

En Unity klasse-fil ser nogenlunde sådan her:

```
1   using UnityEngine;
    0 references
2   public class Mook : MonoBehaviour
3   {
4       private float health;
        0 references
5       void Start()
6       {
7           health = 100;
8       }
        0 references
9       void Update()
10      {
11          if (health > 0)
12          {
13              //search for player
14              //if you encounter the player on the road, kill him
15              //if you get shot, remove a random amount of health
16          }
17      }
18  }
```

PRAKSIS EKSEMPEL

Kort introduktion til Unitys brug af C#

using UnityEngine; –Fortæller C# at vi vil benytte Unitys libraries, der lader os forbinde til Unity spil enginen.

public class Mook : MonoBehaviour { – Denne linje deklarerer klassen og dens navn (“Mook”);

private float health; –Dette erklærer en privat klasse variabel (som kun kan ændres inde fra klassen). Variablen får en værdi i Start ().

void Start () { –Dette deklarerer en metode kaldet "Start." Start er en særlig metode, der kører kun én gang, når spillet første gang startes.

void Update(){ -Update er en anden særlig metode, som kører på hver frame. De fleste af ens spil logik vil blive lagt her.

PRAKSIS

PaddleHandler.cs

```
using UnityEngine;
using System.Collections;

0 references
public class PaddleHandler : MonoBehaviour
{
    //this creates an empty ParticleSystem variable that we can attach to the correct particlesystem in the editor
    public ParticleSystem collision;
    // Use this for initialization
    0 references
    void Start()
    {
    }
    // Update is called once per frame
    0 references
    void Update()
    {
        //this if statement moves the paddle down
        if (transform.position.z > 0 && Input.GetKey(KeyCode.UpArrow))
        {
            float zup = transform.position.z - 16 * Time.deltaTime;
            transform.position = new Vector3(transform.position.x, 0, zup);
        }
        //this if statement moves the paddle up
        if (transform.position.z < 7 && Input.GetKey(KeyCode.DownArrow))
        {
            float zdown = transform.position.z + 16 * Time.deltaTime;
            transform.position = new Vector3(transform.position.x, 0, zdown);
        }
    }
    //this plays the particle effect during a collision
    0 references
    void OnCollisionEnter(Collision other)
    {
        collision.Play();
    }
}
```

LEKTIE

Kig på dette til næste gang

LEKTIE

https://mva.microsoft.com/en-US/training-courses/programming-in-c-jump-start-14254 - 01

Læs

- https://www.tutorialspoint.com/csharp/csharp_data_types.htm til https://www.tutorialspoint.com/csharp/csharp_type_conversion.htm,
- https://www.tutorialspoint.com/csharp/csharp_operators.htm

KILDER

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

KILDER

Data typer

https://www.nemprogrammering.dk/Tutorials/c-sharp/5-strings.php

http://practiceexercisescsharp.blogspot.dk/p/3-basic-data-types-3.html

https://msdn.microsoft.com/en-us/library/aa288453(v=vs.71).aspx

http://practiceexercisescsharp.blogspot.dk/p/4-arrays-structures-and-strings.html

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

Praksis eksempel

http://www.makeuseof.com/tag/programming-game-unity-beginners-guide/

http://www.makeuseof.com/tag/programming-game-unity-beginners-guide/#chapter-10