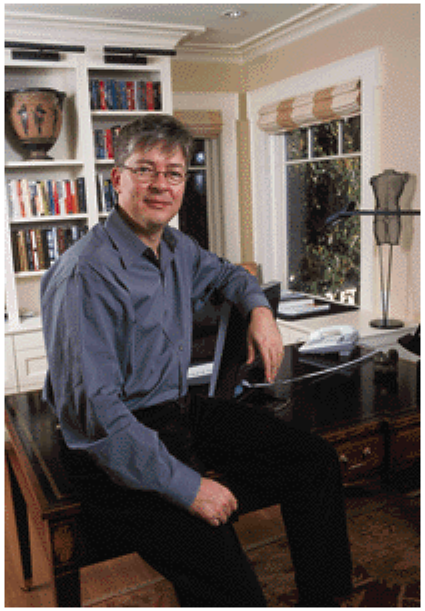


Anders Hejlsberg
Chief architect of C#
Danish engineer (almost) from DIA (DTU).
Creator of Turbo Pascal and Delfi while working for Borland.

C# Basics

Hvor der fokuseres på det nye i sproget som adskiller det fra C++

Mads Torgersen
Language PM for C#.
Before joining Microsoft, Mads worked as an associate professor in computer science at the university of Aarhus, where he was part of the group that developed wildcards for Java generics.

Agenda

- Introduktion til C#
- Egenskaber C# har tilfælles med Java
- Classes
- Data types
- Iterations
- Valgstrukturer
- Exceptions
- Properties
- Indexers
- Interfaces
- Arrays
- Reference types, value types and Boxing
- Operator overload
- Strings in C# and .Net

Læringsmål:
Beskrive og anvende programmeringssproget C#.

Hvorfor lave et nyt sprog ?

Her er Anders Hejlsbergs forklaring på hvorfor Microsoft har lavet et nyt programmeringssprog:

- To produce the first component-oriented language in the C/C++ family
- To create a language in which everything really is an object
- To enable construction of robust and durable software
- **To simplify C++, yet preserve the skills and investment programmers already have**

”To simplify C++, yet preserve the skills and investment programmers already have”

Syntaksen for C# ligner den vi kender fra C++ og Java. Programudviklere vil derfor relativt hurtig blive fortrolig med C#

Fra C++ er er bl.a. fjernet:

- `delete`
- multipel nedarvning
- head’er filer

Hello World

Alt skal ligge inde i en klasse

Programmet starter normalt eksekvering ved funktionen Main (med stort M), men dette kan ændres til et vilkårligt funktionsnavn. Dog skal entry-point funktionen være statisk!

```
class Hello
{
   static void Main()
   {
     System.Console.WriteLine("Hello World");
   }
}
```

Et namespace

En Klasse

En statisk funktion

Ved brug af Visual Studio .NET

```
using System;
namespace ConsoleApp1
{
  /// <summary>
  /// Summary description for Class1.
  /// </summary>
  class Class1
  {
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main(string[] args)
    {
      //
      // TODO: Add code to start application here
      //
      Console.WriteLine("Hello World");
    }
  }
}
```

Åbner et namespace, så man er fri for at skrive dette hver gang.

Placere koden i vores eget namespace. (Du bør vælge et bedre navn end dette!)

XML-tag'et kommentar.

En attribute på funktion Main som gør den COM-safe. I de fleste programmer i I4GUI kan denne slettes uden problemer.

Console Output 1

- Der er 2 funktioner til karakterbaseret udskrivning på skærmen:
  - Write(*string*) findes i 18 varianter (overloads)
  - WriteLine(string) findes i 19 varianter (overloads)

- Og der er grundlæggende 2 måder at bruge dem på:
  - **WriteLine("The sum of " + a + " and " + b + " equals " + c);**

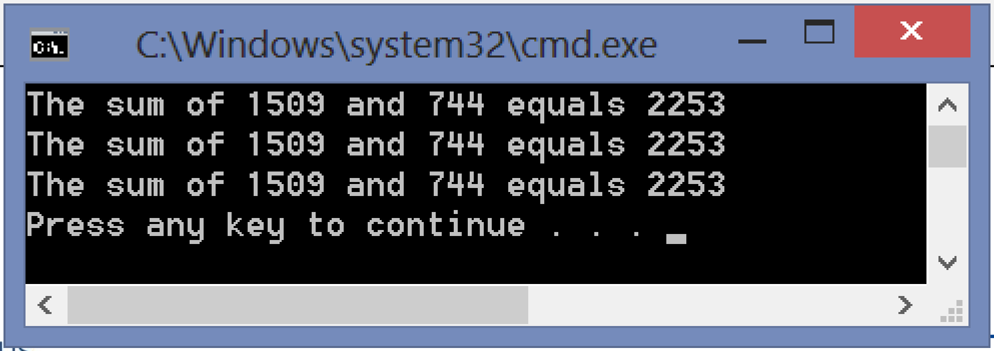
int a konverteres til en streng, som sættes sammen med de øvrige strenge

  - **WriteLine("The sum of {0} and {1} equals {2}", a, b, c);**

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Console Output 2

```
public static void Main()
{
  int a = 1509;
  int b = 744;
  int c = a + b;

  Console.Write("The sum of ");
  Console.Write(a);
  Console.Write(" and ");
  Console.Write(b);
  Console.Write(" equals ");
  Console.WriteLine(c);

  Console.WriteLine("The sum of " + a + " and " + b + " equals "+ c);

  Console.WriteLine("The sum of {0} and {1} equals {2}", a, b, c);
}
```

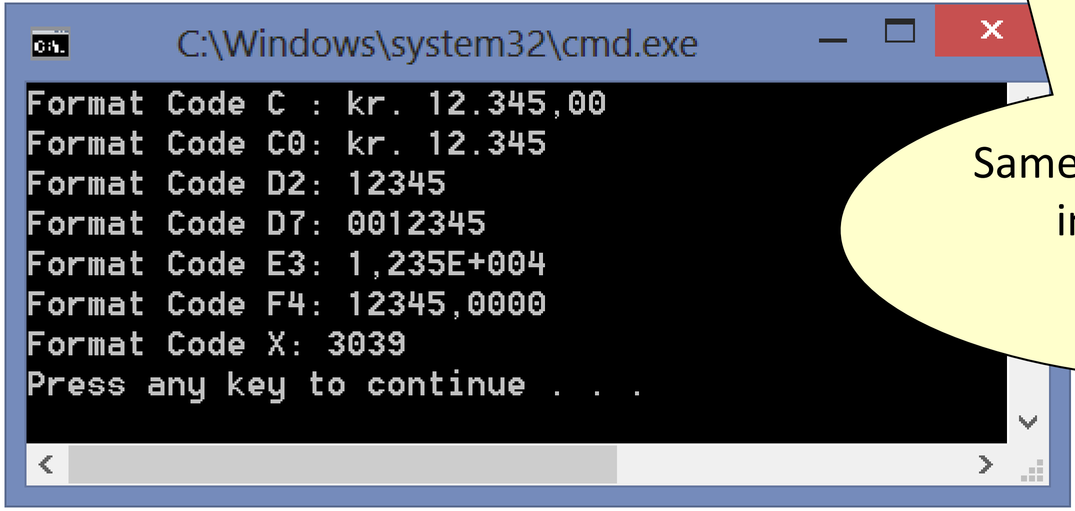


```
The sum of 1509 and 744 equals 2253
The sum of 1509 and 744 equals 2253
The sum of 1509 and 744 equals 2253
Press any key to continue . . .
```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Basic Output Number Formatting

```
static void Main(string[] args)
{
  int tal = 12345;
  Console.WriteLine("Format Code C : {0:C}", tal);
  Console.WriteLine("Format Code C0: {0:C0}", tal);
  Console.WriteLine("Format Code D2: {0:D2}", tal);
  Console.WriteLine("Format Code D7: {0:D7}", tal);
  Console.WriteLine("Format Code E3: {0:E3}", tal);
  Console.WriteLine("Format Code F4: {0:F4}", tal);
  Console.WriteLine("Format Code X: {0:X}", tal);
}
```



```
Format Code C : kr. 12.345,00
Format Code C0: kr. 12.345
Format Code D2: 12345
Format Code D7: 0012345
Format Code E3: 1,235E+004
Format Code F4: 12345,0000
Format Code X: 3039
Press any key to continue . . .
```

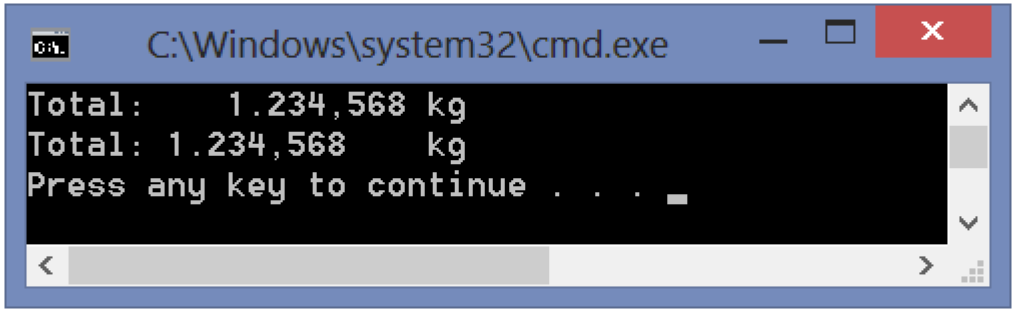
Same technic can be used in string.Format()

Output Formatting - Alignment

- The formal syntax for formatting output is:
  {*index*[,*alignment*][:*formatString*]}

- Examples:

```
double total = 1234.5678;
Console.WriteLine("Total: {0,12:n3} kg", total); //Right aligned
Console.WriteLine("Total: {0,-12:n3} kg", total);//Left aligned
```

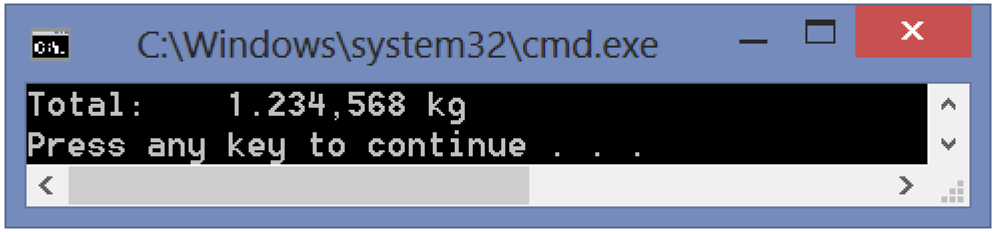


```
Total:     1.234,568 kg
Total: 1.234,568     kg
Press any key to continue . . .
```

Output Formatting With string.Format

- string.Format accepts the same formatting specifiers as Console.WriteLine.

```
double total = 1234.5678;
string str;
str = string.Format("Total: {0,12:n3} kg", total);
Console.WriteLine(str);
```



```
Total:     1.234,568 kg
Press any key to continue . . .
```

You can find many more examples – e.g. Custom number formatting and formating dates, here: http://blog.stevex.net/string-formatting-in-csharp/
And here: http://blog.stevex.net/c-string-formatting-faq/

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Console Input

- Der er også 2 funktioner til karakterbaset input:
  - **static** int **Read();**
    returns the next character from the input stream
  - string **ReadLine();**
    returns the next line from the input stream

```
int i;
char c;
i = Console.Read ();
if (i == -1) break;
c = (char) i;
```

```
Console.WriteLine(“Your name: ");
String name = Console.ReadLine();
```

```
Console.WriteLine(“Type age: ");
String str = Console.ReadLine();
int age = int.Parse(str);
```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

C# og Java side 1

Fælles egenskaber ved C# og Java, som anses for en forbedring fra C++.

- Compiles into machine-independent code which runs in a managed execution environment.
- Garbage Collection coupled with the elimination of pointers
  (in C# restricted use of pointers is permitted within code marked unsafe)
- Powerful reflection capabilities
- No header files, all code scoped to packages or assemblies, no problems declaring one class before another with circular dependencies (inside the same assembly)
- Classes all descend from Object and must be allocated on the heap with new keyword
- Thread support by putting a lock on objects when entering code marked as locked/synchronized

C# og Java side 2

Fælles egenskaber ved C# og Java, som anses for en forbedring fra C++.

- Interfaces, with multiple-inheritance of interfaces, single inheritance of implementations
- Inner classes
- No concept of inheriting a class with a specified access level
- No global functions or constants, everything belongs to a class
- Arrays and strings with lengths built-in and bounds checking
- The "." operator is always used, no more ->, :: operators
- null and boolean/bool are keywords
- All values are initialized before use
- Can't use integers to govern if statements
- try blocks can have a finally clause

Klasser i C#

- Virker næsten som i C++, men indfører nogle nye begreber:

– internal	*(kan kun kaldes fra samme assembly)*
– readonly	(*datafelts værdi kan ikke ændres efter instantiering*)
– properties	(*nyt begreb (kommer fra VB) forklares senere*)
– base	*(kald af basisklassens konstruktør/metode)*
– sealed	*(Der kan ikke nedarves fra denne klasse)*
– abstract	(*nyt keyword til kendt begreb*)
– override, new	*(til versionering)*
– og andre	

Default

Skal skrives hver gang!

```
public class myClass
{
   private int x1;
   private int x2;

   public int GetX1 ()
   {
      return x1;
   }
}
```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

C# Parameter Modifiers

```
public SomeFunction( ? int x) { … };
```

Parameter Modifier	Retning	Betydning
ingen	Ind	Pass by value. Data kopieres på stakken
`out`	Ud	Pass by reference. Kaldte funktion tildeler en værdi til den overførte reference.
`ref`	Ind-ud	Pass by reference. Skal være tildelt en værdi før kald.
`params`		Tillader at der sendes et vilkårligt antal parametre af samme type til funktionen.

Optional Parameters

- The definition of a method or constructor can specify that its parameters are required or that they are optional.
- Optional parameter has a default value as part of its definition.
  - If no argument is sent for that parameter, the default value is used.
  - Default values must be constants.
  - Optional parameters are defined at the end of the parameter list, after any required parameters.

```
public void ExampleMethod(int required,
                          string optionalstr = "default value",
                          int optionalint = 10)
{ .. }
```

  - If the caller provides an argument for any one of a succession of optional parameters, it must provide arguments for all preceding optional parameters. Comma-separated gaps in the argument list are not supported.

Named Parameters

- Named arguments enable you to specify an argument for a particular parameter by associating the argument with the parameter's name rather than with the parameter's position in the parameter list.

```
static double CalculateBMI (double weight, double height)
```

- If you do not remember the order of the parameters but you do know their names, you can send the arguments in either order, weight first or height first.

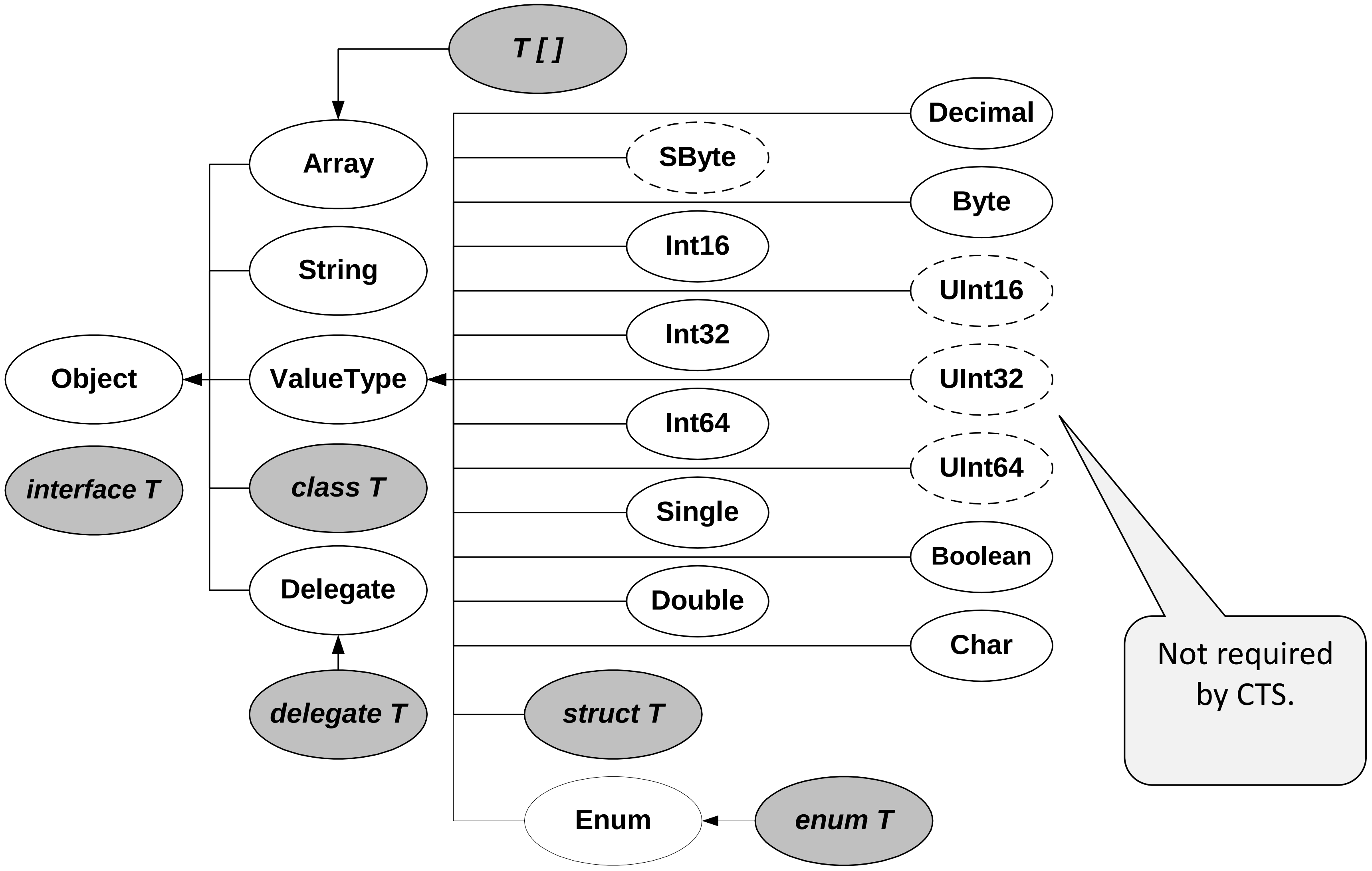
```
CalculateBMI (weight: 87, height: 1.9);
CalculateBMI (height: 1.9, weight: 87);
```

- Positional arguments cannot follow named arguments, but named arguments can follow positional arguments.

Named and Optional Parameters

- Named and optional parameters, when used together, enable you to supply arguments for only a few parameters from a list of optional parameters.
- This capability greatly facilitates calls to COM interfaces such as the Microsoft Office Automation APIs.

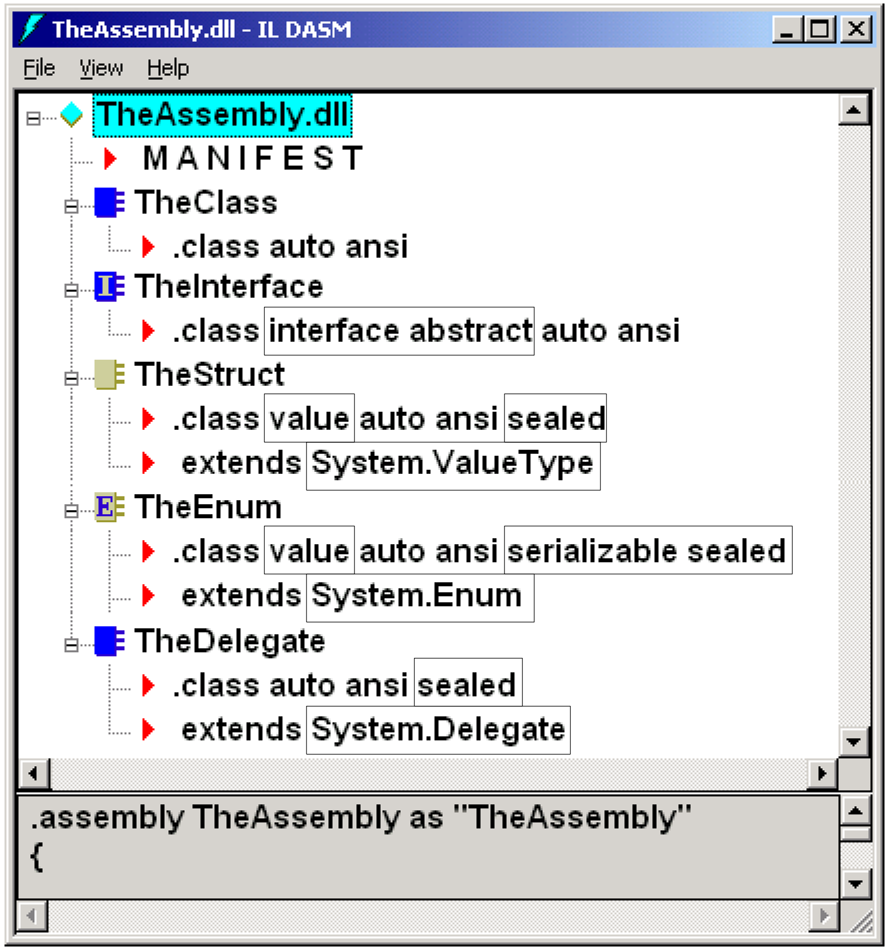
Data types in the CLR
Common Type System CTS



Data types by Language

CLR	C++	C#	VB.NET	Java
Object	(void)	object	Object	object
String	string	string	String	string
Boolean	bool	bool	Boolean	Boolean
Char	char	char	Char	char
Single	float	float	Single	float
Double	double	double	Double	double
Decimal	-	decimal	Decimal	-
SByte	char	sbyte	-	byte
Byte	unsigned char	byte	Byte	-
Int16	short	short	Short	short
UInt16	unsigned short	ushort	-	-
Int32	int	int	Integer	int
UInt32	unsigned int	uint	-	-
Int64	-	long	Long	long
Uint64	-	ulong	-	-

Types in ILDASM



```
TheAssembly.dll
    MANIFEST
    TheClass
        .class auto ansi
    TheInterface
        .class interface abstract auto ansi
    TheStruct
        .class value auto ansi sealed
        extends System.ValueType
    TheEnum
        .class value auto ansi serializable sealed
        extends System.Enum
    TheDelegate
        .class auto ansi sealed
        extends System.Delegate
```

```
.assembly TheAssembly as "TheAssembly"
{
```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Iterations

- C# has 4 different types of iterations:
  - for
  - while
  - do while
  - foreach

```
while (a < 5)
{
    a += 2;
}
```

```
do
{
    a += 2;
} while (a < 5);
```

```
for (i = 0; i < 100; i++)
{
    …
}
```

foreach statement

- Iteration of arrays

```
public static void Main(string[] args)
{
   foreach (string s in args)
     Console.WriteLine(s);
}
```

- Iteration of IEnumerable collections

```
List<Account> accounts = Bank.GetAccounts(...);
foreach (Account a in accounts)
{
   if (a.Balance < 0) Console.WriteLine(a.CustName);
}
```

Hejlsberg: *”To enable construction of robust and durable software”*

C# er helt fra start af designed til at være et robust og pålideligt sprog.

For at opnå dette mål har sproget følgende egenskaber:

- **Garbage collection**
- **Strukturered exception handling**
- **Type sikkerhed**
- **Versionering**

Disse egenskaber fjerner en række fejlkilder som ofte plager C++ programmer.

Exception Handling

Næsten som i C++ men har også mulighed for en

finaly blok

```
try {
// Læg beslag på en ressource, f.eks. en fil
}

catch (SomeExceptionType e) {
// Håndter den konkrete fejlsituation
}

finaly {
// Frigiv ressourcen uanset hvad
}
```

Exceptions virker på tværs af alle sprogene på .Net platformen!

Overflow checking

- Integer arithmetic operations
  - C, C++, Java silently overflow
- checked vs. unchecked contexts
  - Default is unchecked, except for constants
  - Change with “/checked” compiler switch

```
int i = checked(x * y);
```

```
checked {
   int i = x * y;
}
```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Hejlsberg: *"To produce the first component-oriented language in the C/C++ family"*

Programudvikling handler mindre om udvikling af nye store programmer men mere om udvikling af software-komponenter som kan anvendes i mange forskellige sammenhænge.

Typiske egenskaber ved komponenter er:

- **interfaces**
- **properties**
- **methods**
- **events**
- **attributes**

Alle disse begreber er indbyggede i C#, hvilket gør C# til det naturlige valg når der skal udvikles nye komponenter

→ ITKPU

Properties

- Properties are "smart fields"
  - Natural syntax, accessors, inlining

```
public class Button: Control
{
   private string m_caption;

   public string Caption {
      get {
         return m_caption;
      }
      set {
         m_caption = value;
         Repaint();
      }
   }
}
```

```
Button b = new Button();
b.Caption = "OK";
String s = b.Caption;
```

Automatically Implemented Properties

- Allows you to express trivial properties with less code.

```
public class Person
{
  public string Name { get; set; }
}
```

You write

Is compiled as

```
public class Person
{
  private string <Name>k__BackingField;
  public string Name
  {
    get { return <Name>k__BackingField; }
    set { <Name>k__BackingField = value; }
  }
}
```

Initialize Properties with Object Initializer

```
public class Person
{
  public int Age { get; set; }
  public string Name { get; set; }

  public Person() { }
  public Person(string name)
  {
    Name = name;
  }
}
```

- With object initializers you can create an object and initialize all relevant properties in one expression.

```
Person tom1 = new Person();
tom1.Name = "Tom";
tom1.Age = 6;
Person tom2 = new Person("Tom");
tom2.Age = 6;

Person tom3 = new Person() { Name="Tom", Age = 6 };
Person tom4 = new Person { Name="Tom", Age = 6 };
Person tom5 = new Person("Tom") { Age = 6 };
```

Object Initializer

Indexers

- Indexers svarer til overload af []-operatoren i C++
  - Gør at brugerdefinerede klasser kan bruges som et array
  - Kan overloades – dvs. indexering på flere datatyper

```
public class ListBox: Control {
  private string[] items;

  public string this[int index]{
    get {
      return items[index];
      }
    set {
      items[index] = value;
      Repaint();
      }
  }
```

```
ListBox lBox = new ListBox();
lBox[0] = "hello";
Console.WriteLine(lBox[0]);
```

Datatypen Interface

- Et interface er en samling semantisk relaterede abstrakte medlemmer (funktioner eller properties).
- Et interface kan bestå af blot en medlemsfunktion, eller det kan indeholde mange.
- Begrebet interface findes ikke i C++, men svarer til en abstrakt basisklasse uden datamedlemmer og hvor alle medlemsfunktionerne er rent virtuelle.

Eksempel på definition

```
// The pointy behavior.
public interface IPointy
{
            // an interface contains functions
           byte GetNumberOfPoints();



          // an interface can also have properties...
          byte Points{get; set;}
          // but can't have datamembers
}
```

Er implicit abstract

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Implementering af et interface

```
public class Triangle : Shape, IPointy
{
        public Triangle(string name): base(name)
        {}
        public Triangle()
        {}
        public override void Draw()
        {
                Console.WriteLine("Drawing " + PetName + " the Triangle");
        }
        // IPointy interface
        public byte GetNumberOfPoints()
        {
                return 3;
        }
}
```

Interface reference 1

Der 3 måder at lave en reference til et interface på:

Den langsomme: Typecast med exceptionhandling

```
Triangle t = new Triangle();
IPointy itfPt;
try
{
    itfPt = (IPointy)t;
    ...
}
Catch(InvalidCastException e)
{Console.WriteLine("OOPS!  Not pointy...");}
```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Interface reference 2 - **as**

Brug af **as** operatoren:

```
Triangle t = new Triangle();
IPointy itfPt;
itfPt = t as IPointy;
if (itfPt != null)
      Console.WriteLine("Got interface using
      as keyword");
else
      Console.WriteLine("OOPS!  Not
            pointy...");
```

Note:
The **as** *operator can only be used on reference types!*
And it is ”best practice” to use the as operator when you need to type cast a reference type (when you can).

Interface reference 3 - **is**

Brug af **is** operatoren:

```
Triangle t = new Triangle();
IPointy itfPt;

if(t is IPointy)
  {
   itfPt = (IPointy)t;
   ...
  }
else
 Console.WriteLine("OOPS!  Not pointy...");
```

Note:
The **`is`** *operator can also be used on value types!*
And it is ”best practice” to use the is operator together with normal type cast on value types (when you can).

Brug af Interfaces

```
Shape[] s = {new Hexagon(), new Circle(),
                 new Triangle("Joe"), new Circle("JoJo")};

For (int i = 0; i < s.Length; i++)
{
                s[i].Draw();

                // Who's pointy?
                if (s[i] is IPointy)
                    Console.WriteLine("Points: {0}\n",
                                  ((IPointy)s[i]).GetNumberOfPoints());
```

Hvorfor ?

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Interface som parameter

```
// The 3D drawing behavior.
public interface IDraw3D
{
        void Draw();
}
// I'll Draw anything!
public static void DrawThisShapeIn3D(IDraw3D itf3d)
{
        itf3d.Draw();
}
```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Explicit Interface Implementation

- Bruges for at undgå navne sammenfald.
- Bruges for at skjule metoderne i et (eller flere) interface for brugeren af objekter af klassen.

```
public class Circle : Shape, IDraw3D
{
        public override void Draw()

        void IDraw3D.Draw()
        {
                Console.WriteLine("Drawing Circle in 3D!");
        }
```

Kan kun kaldes via en interface reference

Brug af interfaces i .Net

- .Net frameworket indeholder mange interface definitioner, som vi udviklere kan vælge at implementere for derved at for adgang til noget specifikt funktionalitet i frameworket
- Programmering til interfaces er meget vigtigt for de fleste Windows-programmører

IEnumerable og IEnumerator

- IEnumerable
```
public interface IEnumerable
{
    public IEnumerator GetEnumerator();
}
```

-
  ```
  public interface IEnumerator
  {
      bool MoveNext ();
      object Current {get;}
      void Reset ();
  }
  ```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Arrays in C#

- Declaring an array:
  - `int[] A;`
- Allocating memory for the array requires a *new* expression
  - `A = new int[100];`
- Arrays are Zero-based
  - The first element is always at index 0
- Use of array:

  -
    ```
    for (int i = 0; i < 100; i++)
       A[i] = i * i;
    ```
- Bounds checking
  - `int x = A[7];   // ok`
  - `int y = A[100]; // Throws out-of-bounds exception`

Arrays

- Any array is implicitly an object of type *System.Array*, and you can use properties and methods defined by the *Array* class with arrays.
  - `A.Length` is the value 100.

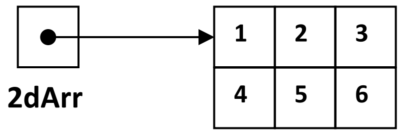
- The number of elements in an array can be determined at runtime:

  -
    ```
    Console.Write("Enter the array size: ");
    int num = Int32.Parse(Console.ReadLine());
    double[] D = new double[num];
    ```
  - But you cannot change the size of an Array once it is initialized (*use one of the collection classes if you need to do that*).

- Initialize

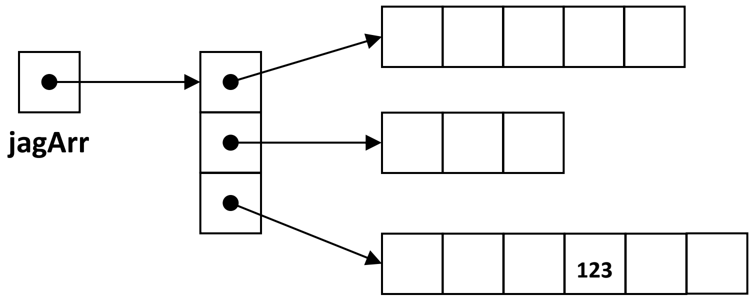
  -
    ```
    int[] prime = new int[] {1,2,3,5,7,11,13,17,19};
    int[] prime = {1,2,3,5,7,11,13,17,19};
    ```

Multidimensional Arrays



- Rectangular
  - `int[,] 2dArr = new int[2,3];`
  - Can initialize declaratively
  - `int[,] 2dArr = new int[2,3] { {1,2,3}, {4,5,6} };`

- Jagged
  - An array of arrays
  - `int[][] jagArr = new int[3][];`
  - `jagArr[0] = new int[5];`
  - `jagArr[1] = new int[3];`
  - `jagArr[2] = new int[6];`

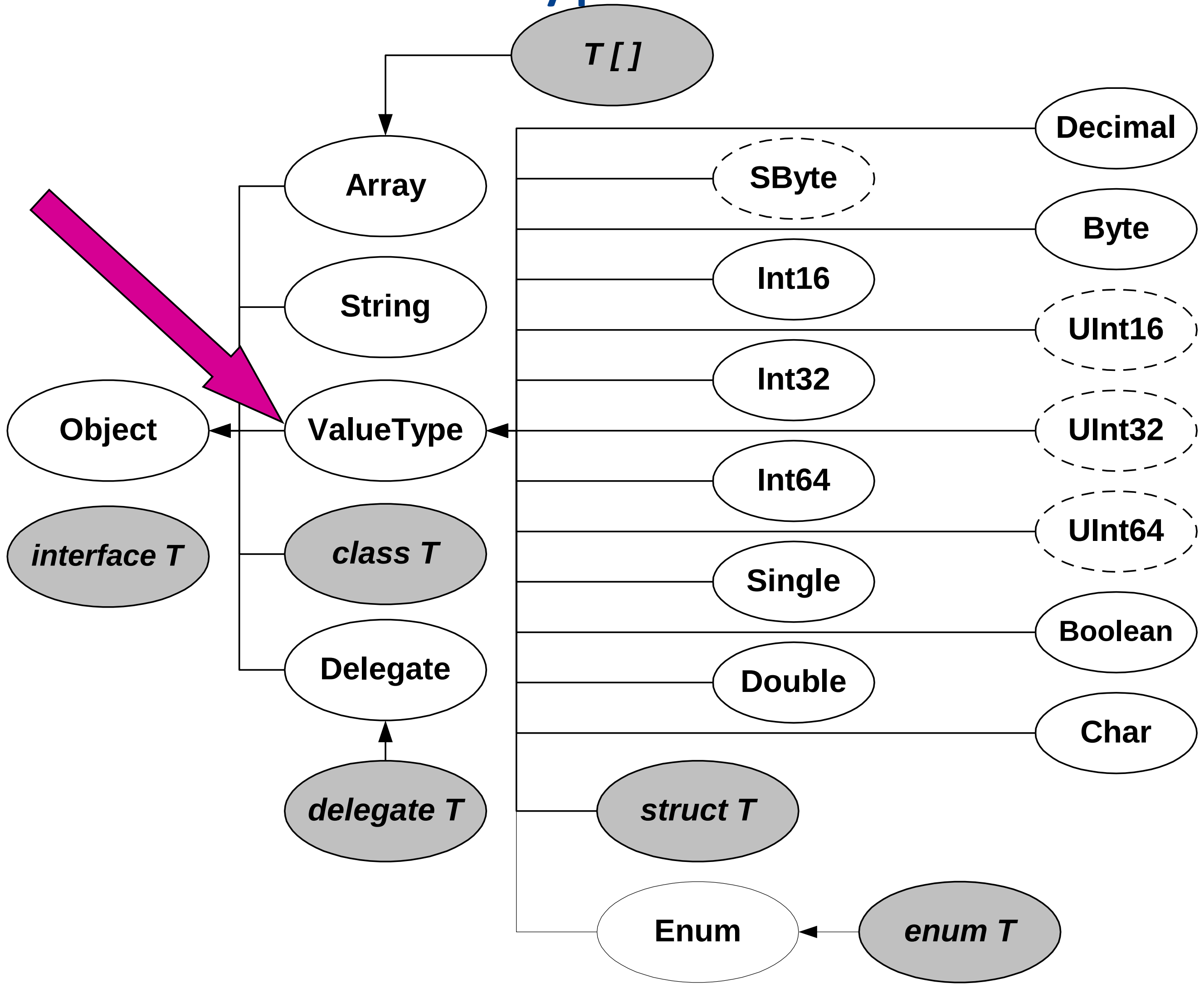


  - Must initialize procedurally
  - `jagArr[2][3] = 123;`

REFERENCE AND VALUS TYPES

En hurtig gennemgang af forskellen på reference typer og value typer samt boxing

Datatyper i CLR

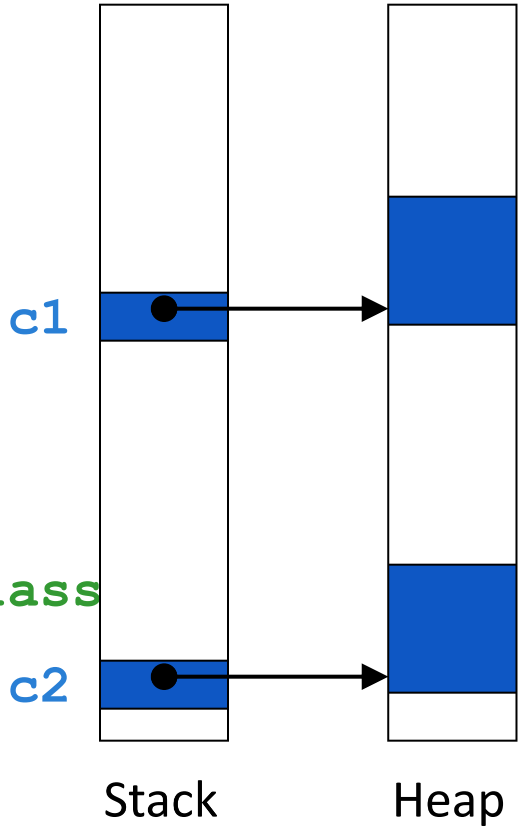


▨

All gray types are UDT (user defined types)

Instantiering af ”reference” objekter

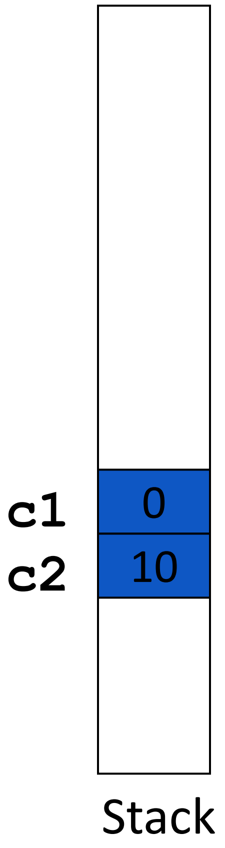
```
class HelloClass
{  public void SayHi() {...}
}

class HelloApp
{
  public static int Main(string[] args)
    {
    // Make instance of HelloClass.
    HelloClass c1 = new HelloClass();
    c1.SayHi();

    // Make another instance of HelloClass
    HelloClass c2;
    c2 = new HelloClass();
    ...
    }
}
```



AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Instantiering af ”value” objekter

```
struct HelloStruct
{
  public void SayHi() {...};
  public int x;
}

class HelloApp
{
  public static int Main(string[] args)
    {
    // Make instance of HelloStruct.
    HelloStruct c1 = new HelloStruct ();
    c1.SayHi();
    // Make another instance of HelloStruct.
    HelloStruct c2; // Use of new is optional
    c2.x = 10;
    c2.SayHi();
}
```

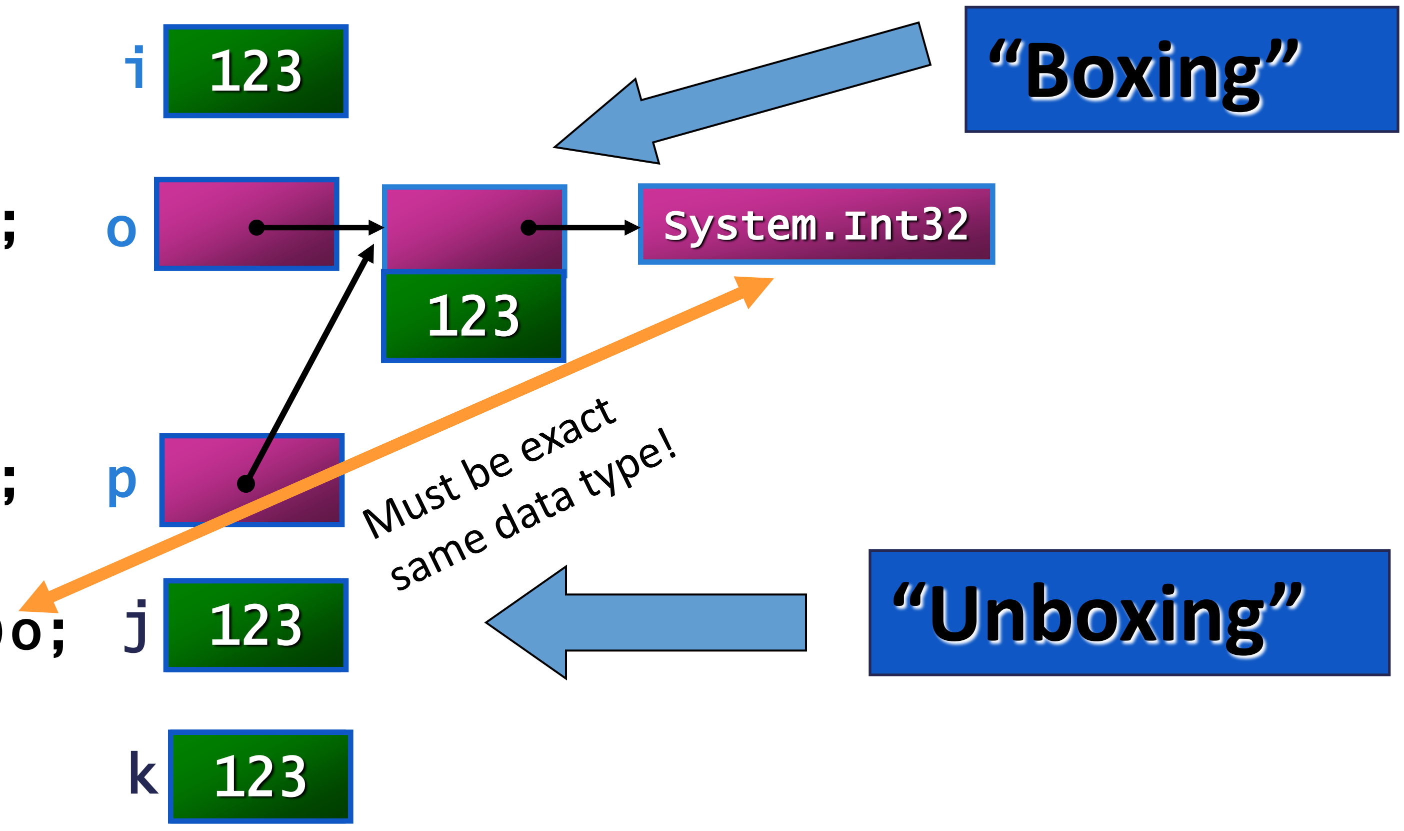


AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Unified Type System

“Everything is an object”

```
int i = 123;

object o = i;

object p = o;

int j = (int)o;

int k = j;

Console.Writeline(i.ToString());
```



Hvorfor Boxing ?

”To create a language in wich everything really is an object”

Med introduktionen af begrebet ”boxing” og ”unboxing” bygger C# bro mellem de primitive typer og klasse-typerne, hvorved man opnår at alle typer af data kan behandles som et objekt.

Endvidere introducerer C# begrebet ”value types” som allokeres på stacken, hvilket i mange sammenhænge kræver mindre plads i hukommelsen og giver hurtigere programafvikling.

Reference contra value types

- Reference types yield "true" objects
- Value types yield formatted memory
- In C# and VB, all classes are reference types
- In C# and VB, primitives, structs and enums are value types
- Instances of value types do not have an object header
- Instances of value types are not independently garbage collected

Operator Overload

- Mange operatorer (men ikke alle) kan overloades for brugerdefinerede klasser og structs.
- Gør at man kan få brugerdefinerede datatyper til at fremstå som om de var indbyggede i sproget
- Er altid statiske funktioner
- Kan kun laves til eller fra en brugerdefineret type, eller mellem 2 brugerdefinerede typer

```
public struct Matrix {
  public static Matrix operator * (Matrix left, Matrix
right)
    {
          return new Matrix (…
    }
...
}
```

```
Matrix a, b, c;
…
a = b * c;
```

Brugerdefineret typekonvertering

- Syntaks som ved operatoroverload – blot er der ingen operator.
- implecit indikerer at der er en sikker konvertering hvor der ikke tabes datanøjagtighed eller kastes exceptions.

```
public struct Rational {
   …
   public static implicit operator Rational (int i)
   {
      return new Rational(i,1);
   }
}
```

Til type

Fra type

```
Rational r = 2;
```

- explecit indikerer at der er en usikker konvertering, hvor der enten kan tabes datanøjagtighed og/eller der kan kastes en exceptions.

```
public struct Rational {
   …
   public static explicit operator int (Rational)
   {
      return (numerator / denominator);
   }
}
```

```
Rational r = …;
int n = (int) r;
```

Operator Overload
Rational Number (1/3) som class

```
public class Rational
{
    public Rational(int n, int d)  { … }
}
```

- Heap allocated
- Can be null
- “=” assigns reference not value
- Arrays allocates references not values
  - Rational[] array = new Rational[100];
- **Doesn’t “feel” like a built-in number type!**

Operator Overload
Rational Number (1/3) som struct

```
public struct Rational
{
    public Rational(int n, int d)  { … }
}
```

- Not heap allocated
- Can’t be null
- “=” assigns value
- Arrays allocates values
  - Rational[] numbers = new Rational[100];
- **“feels” like a built-in number type**

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Rational Number

Uden operator-overload og typekonvertering:

```
Rational r1 = new Rational(1,2);
Rational r2 = new Rational(2,1);

Rational r3 = r1.AddRational(r2);

double d = Rational.ConvertToDouble(r3);
```

Med operator-overload og typekonvertering:

```
Rational r1 = new Rational(1,2);
Rational r2 = 2;

Rational r3 = r1 + r2;

double d = (double) r3;
```

STRINGS
IN
C#

char

C# datatypen char er et alias for BCL’s System.Char

char er en value datatype, men kan ikke bruges til beregninger

```
char ch = 'H';
ch += 2;           // Ikke lovligt!
int i = (int) ch;
i += 2;
```

Som i C++ er '\' en speciel escape karekter, f.eks. `'\n'`

char datatypen fylder 2 byte, og karaktertabellen er Unicode

```
ch = '\u03B1';   // ~ 'α'
```

4 cifret hexadecimal karakterkode

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

String

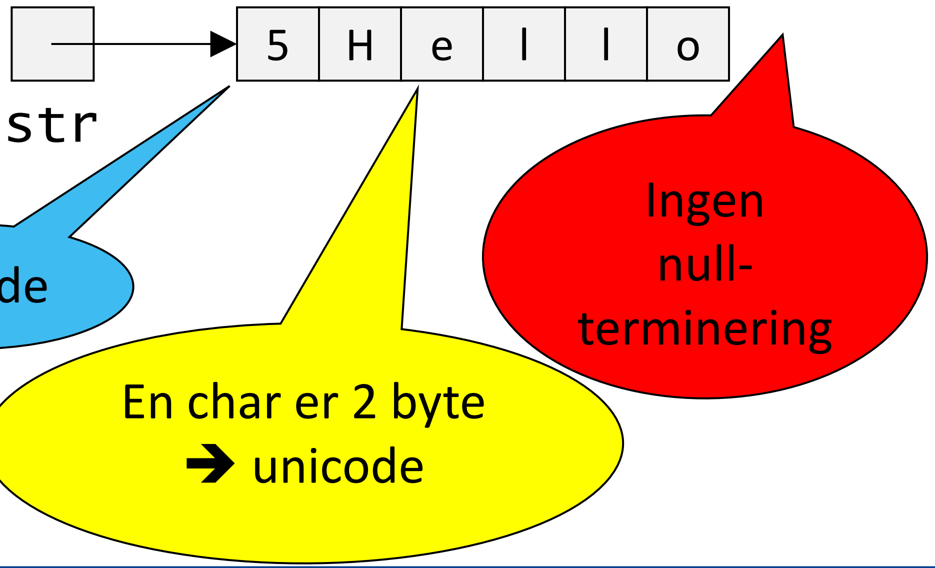
C# datatypen string er et alias for BCL's System.String

string er en indbygget primitive type i alle .NET programmeringssprog, og er direkte supporteret af CIL

string er ikke det samme som et array of char, men der findes funktioner som kan konvertere imellem de 2 typer

string er en reference datatype

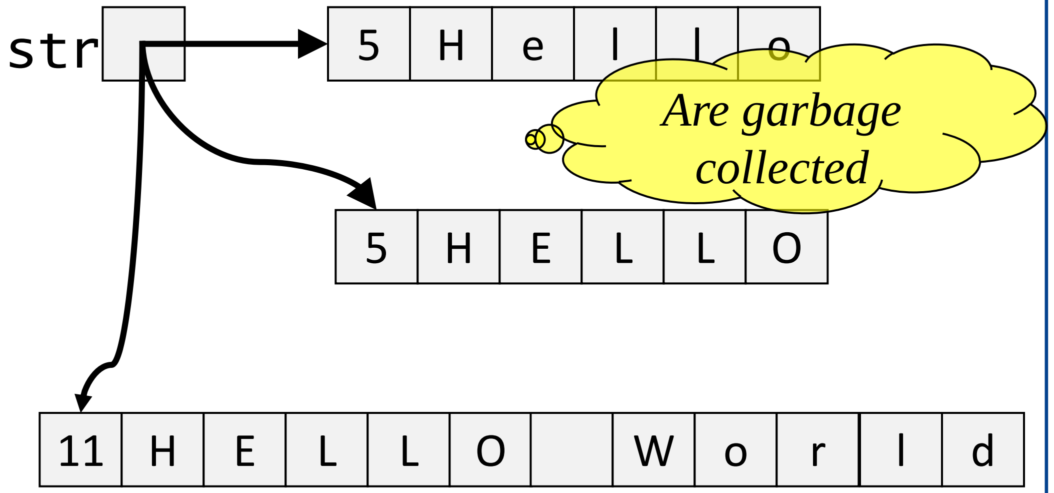
```
string str = "Hello";
```



String is immutable

- Instances represent a single immutable string of characters in memory.
- Characters cannot change.
- Methods on `System.String` return a new instance:
  - `ToUpper()`, `Substring()`, `Split()`, etc.
- String concatenation expressions create new instances.

```
string str = "Hello";
char ch = str[3];
~~str[3] = 'w';~~
~~str.ToUpper();~~
str = str.ToUpper();
str = str + " World";
```



Switch on string

- The switch statement support switching on strings.

```
Color ColorFromFruit(string s) {
   switch (s.ToLower()) {
      case "apple":
         return Color.Red;
      case "banana":
         return Color.Yellow;
      case "carrot":
         return Color.Orange;
      default:
         throw new InvalidArgumentException();
   }
}
```

System.Text.StringBuilder

- Maintains an internal growable buffer
- Random read-write access to characters
- `ToString()` returns a `String` instance that uses `StringBuilder` object’s buffer
- Very efficient!
  - Use `String` for static strings
  - Use `StringBuilder` to create dynamic `String`’s
- **The way to incrementally build instances of `System.String`**

StringWriter

- StrigWriter klassen fra namespaced System.IO kan også bruges til at opbygge en lang streng på en effektiv måde.
- StringWriter har som medlemsfunktioner diverse overloads af Write() og WriteLine()
  Der virker som man forventer, blot lagres karaktererne i en buffer, som efterfølgende kan kopieres til en String

```
StringWriter sw = new StringWriter();

for (int i = 0; i < 100000; i++)
  sw.WriteLine("A part of a dam long string ");

string str = sw.ToString();
```

Regex

Sting klassen har mange medlemsfunktioner og et par properties som gør det let at arbejde med tekststrenge i C#
(læs mere om disse i online-hjælpen).

Men når det drejer sig om de mere avancerede operationer på strenge, som f.eks. at parse dem ud fra en bestemt syntaks, så findes der en anden klasse, som man ofte med fordel kan benytte:

Regex

fra namespaced: System.Text.RegularExpressions

Bemærk, at denne klasse har så mange faciliteter, at der er skrevet en hel bog om kun denne klasse (men man kan godt nøjes med at bruge kun et lille hjørne af dens muligheder)

Implicit Typing of Local Variables

- You can ask the compiler to **infer** the types of local variables.
- Just replace the type part of a normal local variable declaration with var:

```
MyType myVariable1 = GetNewMyType();

var myVariable2 = GetNewMyType();

var variable3 = 0; // Won’t compile
```

The compiler sets the type of myVariable2 to the type returned from GetNewMyType()

myVariable2 is still statically typed!

No type information so compiler can’t infer the type!

References

- C# Programming Guide
  http://msdn.microsoft.com/en-us/library/67ef8sbd.aspx
- C# Language Specification
  http://msdn.microsoft.com/en-us/library/ms228593.aspx

  http://www.microsoft.com/en-us/download/details.aspx?id=7029
- .Net output formatting
  http://blog.stevex.net/string-formatting-in-csharp/
  http://blog.stevex.net/c-string-formatting-faq/