Module 02

"Value Types and Reference Types"

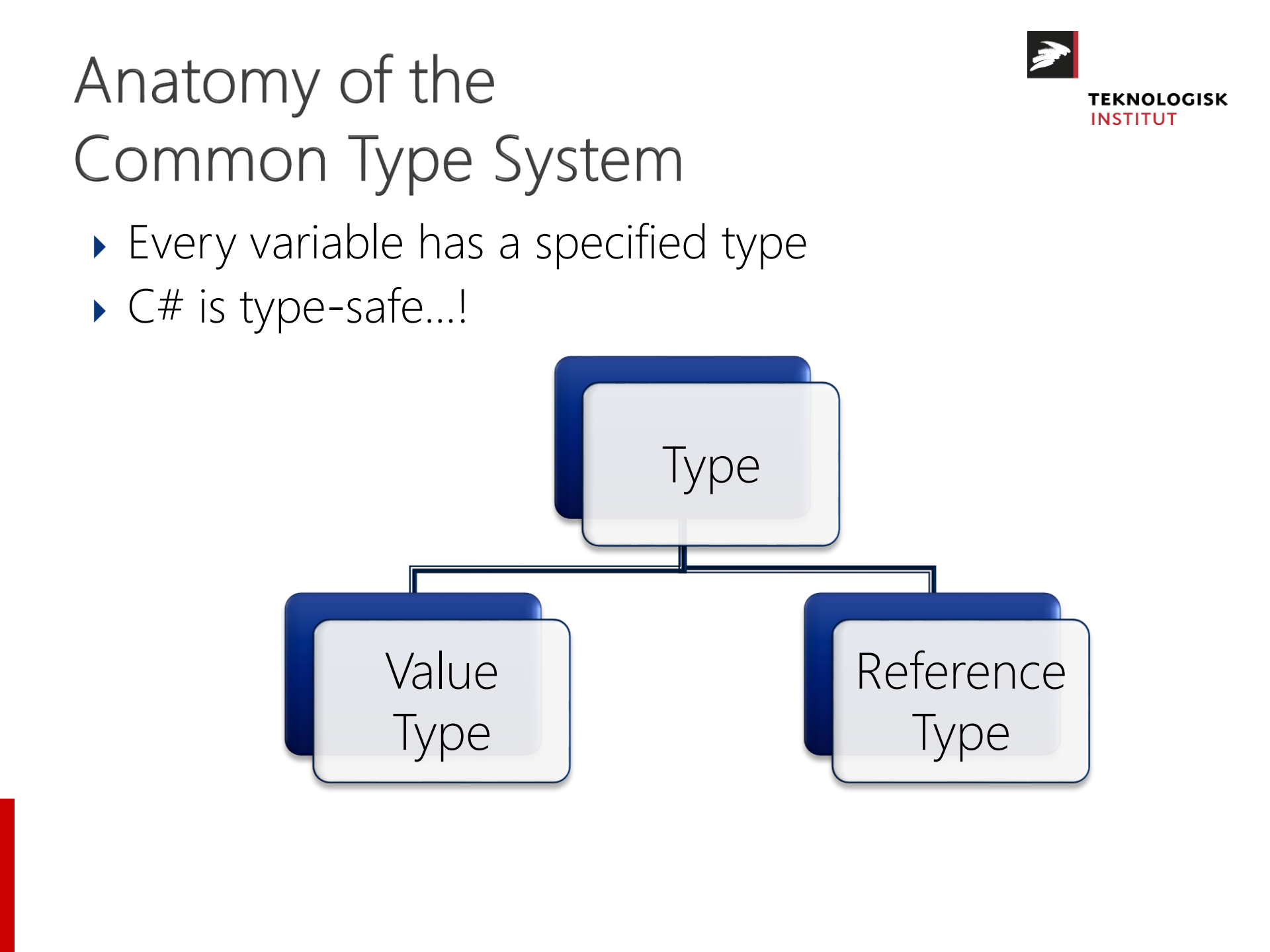
TEKNOLOGISK INSTITUT

TEKNOLOGISK
INSTITUT

Agenda

- .NET Common Type System
- Predefined Value Types
- Expressions
- Data Type Conversions
- User-defined Value Types
- Arrays
- Strings
- Value Types Revisited – `Nullable`

Anatomy of the Common Type System



- Every variable has a specified type
- C# is type-safe…!

Value Types vs. Reference Types

TEKNOLOGISK INSTITUT

Value Types

- Directly contain data
- Allocated on the stack
- Have to be initialized
- Each copy has its own data

Reference Types

- Store references to data ("objects")
- Stored on the heap
- Has a default value of null
- Several references can refer to same data

TEKNOLOGISK
INSTITUT

Agenda

- .NET Common Type System
- **Predefined Value Types**
- Expressions
- Data Type Conversions
- User-defined Value Types
- Arrays
- Strings
- Value Types Revisited – `Nullable`

Overview of Predefined Value Types

TEKNOLOGISK INSTITUT

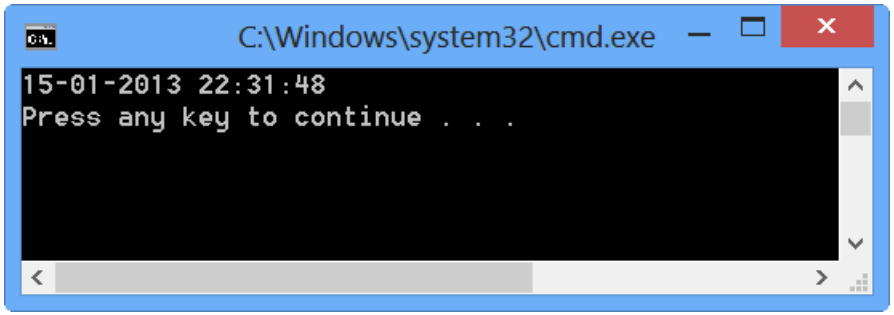
C# Data Type	CTS Type	Description
`bool`	`System.Boolean`	True or false values
`int`	`System.Int32`	Signed integers
`short`	`System.Int16`	Signed short integers
`long`	`System.Int64`	Signed long integers
`uint`	`System.UInt32`	Unsigned integers
`char`	`System.Char`	Character values
`float`	`System.Single`	Single-precision floating
`double`	`System.Double`	Double-precision floating
`decimal`	`System.Decimal`	128-bit precision number

System.DateTime

TEKNOLOGISK INSTITUT

- A very important type with no C# keyword
- Has a lot of interesting details and features
  - High-precision
  - Versatile formatting is built-in

```
Console.WriteLine( DateTime.Now );
```



- A corresponding **System.TimeSpan** also exists

`System.Numerics` Namespace

TEKNOLOGISK INSTITUT

- .NET 4.5 has a **`System.Numerics`** namespace containing
  - **`BigInteger`**
  - **`Complex`**
- These are immutable
- Probably not on the exam!
  - [Troelsen, p. 93 – 95] has more info

TEKNOLOGISK
INSTITUT

Agenda

- .NET Common Type System
- Predefined Value Types
- **Expressions**
- Data Type Conversions
- User-defined Value Types
- Arrays
- Strings
- Value Types Revisited – `Nullable`

TEKNOLOGISK
INSTITUT

Declaring Variables

- Declare by data type and variable name

```
bool isStarted;
```

- Multiple variables can be declared simultaneously

```
int favoriteNumber, i, j;
```

- Local variables can be declared everywhere in methods
- Class-level variables are called *members*

Assigning Values

TEKNOLOGISK INSTITUT

- Assign values by using the assignment operator **=**

```
bool isStarted;
isStarted = true;
```

- Variables can be declared and assigned simultaneously

```
int favoriteNumber = 87, i = 0, j = i;
```

```
char c = 'A';
```

- Note: Variables must be initialized before use!

TEKNOLOGISK
INSTITUT

Naming of Variables

- Compiler requires that only letters, digits, and underscores are used
- C# keywords are reserved
- Case-sensitive!

- Recommendations
  - Don't abbreviate. Characters are cheap! ☺
  - Use camelCasing for variables
  - Use PascalCasing for types, classes, methods.

TEKNOLOGISK
INSTITUT

Constants

- Use the const keyword to declare constants

```
const int favoriteNumber = 87;
```

- Must be initialized when declared

TEKNOLOGISK
INSTITUT

Operators

Operator Type	Operators
Equality	`==` `!=`
Relational	`<` `<=` `>` `>=` `is`
Conditional	`||` `&&` `?:`
Arithmetic	`+` `-` `*` `/` `%`
Increment, Decrement	`++` `--`
Assignment	`=` `+=` `-=` `*=` `/=`

TEKNOLOGISK INSTITUT

Operator Precedence

- Operator precedence determines order of evaluation
  - Multiplicative > Additive.

```
x = y + 87 * z
```

```
x = y + ( 87 * z )
```

- Associativity
  - All binary (except assignment) is left-associative

```
x + y + z
```

```
( x + y ) + z
```

```
a = b = c
```

```
a = ( b = c )
```

- Use parentheses whenever there is doubt!

TEKNOLOGISK
INSTITUT

Agenda

- .NET Common Type System
- Predefined Value Types
- Expressions
- **Data Type Conversions**
- User-defined Value Types
- Arrays
- Strings
- Value Types Revisited – `Nullable`

TEKNOLOGISK
INSTITUT

Implicit Conversions

- Also known as “widening” conversions
- Never lose precision or value

```
short i = 16384;

// Implicit/Widening conversion
int j = i;
```

- Are always allowed by the compiler
- Always succeeds

TEKNOLOGISK
INSTITUT

Explicit Conversions

- Also known as "narrowing" conversions
- Can lose precision or value

```
int i = int.MaxValue;

// Explicit/Narrowing conversion
short j = (short) i;
```

- Are allowed by the compiler
- Might fail!

Overflow Checking

TEKNOLOGISK INSTITUT

- The **checked** keyword turns on over/underflow checking

```
checked
{
    short j = (short) i;
}
```

```
short j = checked( (short) i );
```

- There is an corresponding **unchecked** keyword

- Default (un)checking can be set in the Visual Studio 2012 project properties
  - "Build" -> "Advanced..."

Implicitly Typed Variables

TEKNOLOGISK INSTITUT

- You can define local implicitly typed variables using the **var** keyword

```
var myInteger = 87;
var myBoolean = true;
var myString = "Hello, there...";
```

- The compiler infers the type of the local variable!
- Everything is still completely type-safe

```
var i = 87;
i = 112;
int j = i + 42;
i = "Forbidden!";
```

- Must be assigned a value when declared

```
var myInteger;
myInteger = 87;
```

TEKNOLOGISK
INSTITUT

Agenda

- .NET Common Type System
- Predefined Value Types
- Expressions
- Data Type Conversions
- **User-defined Value Types**
- Arrays
- Strings
- Value Types Revisited – `Nullable`

Enumerations

TEKNOLOGISK INSTITUT

- Used for creating a set of symbolic names

```
enum Fruit
{
    Apple,
    Banana,
    Orange
}
```

```
Fruit f = Fruit.Banana;
```

- Ordering does not have to be sequential – and can also be bit flags!
- Underlying enumeration type can be explicitly chosen

```
enum Team : byte
{
    AGF = 1,
    Brøndby = 6,
    FCK = 5,
    Randers = 12
}
```

```
Team t = Team.AGF;
Console.WriteLine( t ); // ???
```

Structures

TEKNOLOGISK INSTITUT

- Used for defining a structured value consisting of several subvalues

```
struct Point
{
   public int x, y;
}
```

```
Point pt;
pt.x = 42;
Console.WriteLine( pt.y ); // Oops!
```

- Members are private by default
- All subvalues must be initialized before use!

- Value can be default initialized using the **new** construct

```
Point pt = new Point();
Console.WriteLine( pt.y ); // ???
```

TEKNOLOGISK
INSTITUT

Agenda

- .NET Common Type System
- Predefined Value Types
- Expressions
- Data Type Conversions
- User-defined Value Types
- **Arrays**
- Strings
- Value Types Revisited – `Nullable`

What Are Arrays?

TEKNOLOGISK
INSTITUT

- An array is a set of data items

42	87	112	...	256

- All items are of the same type
- An array is accessed using a numerical index starting from 0!

TEKNOLOGISK
INSTITUT

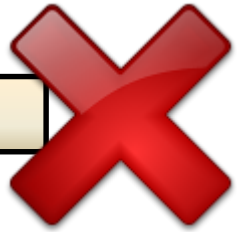
Declaring an Array

- An array variable is declared as

```
Type[] Name;
```

- Array size is not a part of the declaration!

```
int[ 10 ] myArray;
```



- Can declare arrays of several dimensions

```
char[ , ] myCharGrid;
```

```
double[ , , ] myCube;
```

TEKNOLOGISK
INSTITUT

Indexing Arrays

- Arrays are indexed by variable name and index

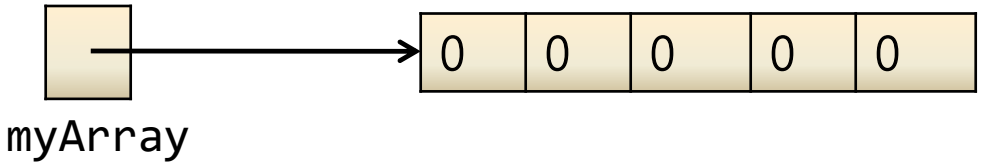
```
int[] myArray;
...
Console.WriteLine( myArray[2] ); // 112
```

Creating Arrays

TEKNOLOGISK INSTITUT

- Declaring an array variable does not create the array itself!
- It must be explicitly created with the **new** operator

```
int[] myArray;
...
myArray = new int[ 5 ];
```




```
int[] myArray = new int[ 5 ];
```

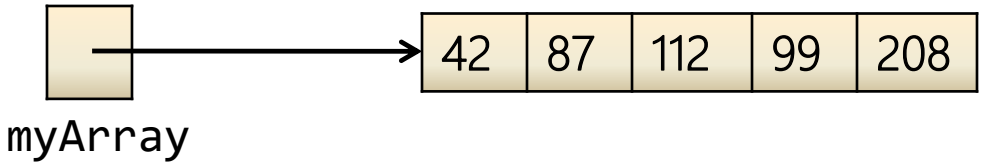
- Arrays are by default initialized with “Zero Whitewash”

Initializing Arrays

TEKNOLOGISK INSTITUT

- Arrays can be explicitly initialized

```
int[] myArray = new int[ 5 ] { 42, 87, 112, 99, 208 };
```



- A convenient shorter syntax exists

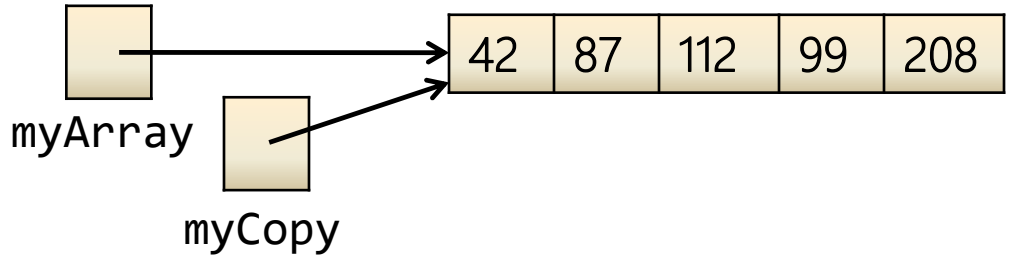
```
int[] myArray = { 42, 87, 112, 99, 208 };
```

Assigning Array Variables

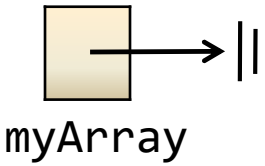
TEKNOLOGISK INSTITUT

- Copying array variables amounts to copying references only!

```
int[] myArray = new int[ 5 ] { 42, 87, 112, 99, 208 };
int[] myCopy = myArray;
myArray[ 1 ] = 0;
Console.WriteLine( myCopy[ 1 ] );
```




```
myArray = null;
```



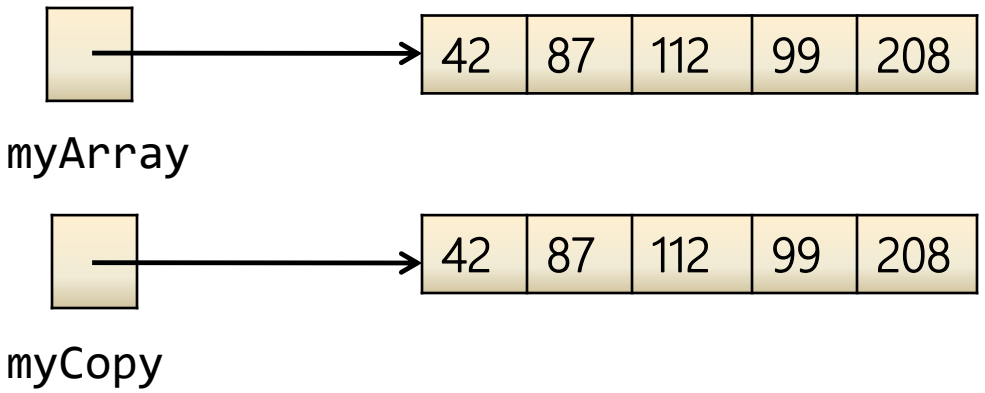
- This is the case for reference types in general

Comparing Array Variables

TEKNOLOGISK INSTITUT

- Comparing array variables amounts to comparing references

```
int[] myArray = { 42, 87, 112, 99, 208 };
int[] myCopy = { 42, 87, 112, 99, 208 };
Console.WriteLine( myArray == myCopy ); // ???
```



- This is the case for reference types in general

TEKNOLOGISK INSTITUT

Implicitly Typed Local Arrays

- Can use implicit typing for arrays

```
var a = new[] { 1, 2, 3, 4 };
var b = new[] { false, true, true };
var c = new[] { "Implicit", "Typing", "Rocks" };
```

- But types cannot be mixed!

```
var d = new[] { 1, "two", 3, false };
```

- In general, the **`var`** keyword can be used with any reference type

```
var o = new Car();
...
o = null;
```

- However, the variable must be non-null upon declaration!

```
var ohNo = null;
```

TEKNOLOGISK
INSTITUT

Array Properties

- Length
- Rank (a.k.a. "dimensions")

```
int[] myArray = new int[ 5 ] { 42, 87, 112, 99, 208 };
Console.WriteLine( myArray.Length ); // 5
Console.WriteLine( myArray.Rank );   // 1
```

```
int[ , ] myGrid = new int[ 2, 3 ] { { 0, 1, 2 }, { 3, 4, 5 } };
Console.WriteLine( myGrid.Length ); // 6
Console.WriteLine( myGrid.Rank );   // 2
```

- Once an array has been created it cannot be resized!

System.Array

TEKNOLOGISK
INSTITUT

- Arrays are instances of **`System.Array`**

- Static methods
  - `Clear()`
  - `Reverse()`
  - **`Sort()`**
  - `IndexOf()`
  - …

TEKNOLOGISK
INSTITUT

Agenda

- .NET Common Type System
- Predefined Value Types
- Expressions
- Data Type Conversions
- User-defined Value Types
- Arrays
- **Strings**
- Value Types Revisited – `Nullable`

System.String

TEKNOLOGISK INSTITUT

- Strings have a number of useful methods and properties
  - `Length`
  - `Compare()`
  - `Contains()`
  - `Equals()`
  - **`Format()`**
  - `Insert()`
  - `PadLeft()`
  - `PadRight()`
  - `Remove()`
  - `Replace()`
  - `Split()`
  - `Substring()`
  - `Trim()`
  - `ToUpper()`
  - `ToLower()`
  - …

Manipulating Strings

TEKNOLOGISK INSTITUT

- The **+** operator concatenates strings

```
string s1 = "Programming ";
string s2 = "C# 5.0";
string s3 = s1 + " in " + s2;
Console.WriteLine( s3 );
```

- It is a convenient shorthand for **String.Concat**

```
string s3 = string.Concat( s1, string.Concat( " in ", s2 ) );
```

- Escaped strings

```
string s = "This is a \t \\tab\\ with newline\r\n";
```

- Verbatim strings

```
string s = @"This is a \t \\tab\\ with newline\r\n";
```

Strings and Equality

TEKNOLOGISK
INSTITUT

- String is a reference type!

```
string s1 = "Hello!";
string s2 = "Hello!";
string t = "Yo!";

Console.WriteLine( s1 == s2 );                    // ???
Console.WriteLine( s1 == "Hello!" );              // ???
Console.WriteLine( s1 == "HELLO!" );              // ???
Console.WriteLine( s1.ToUpper() == "HELLO!" ); // ???
Console.WriteLine( s1.Equals( t ) );              // ???
Console.WriteLine( "Yo!".Equals( t ) );           // ???
```

- The **==** operator has been redefined for strings to compare values
  - Uses the **Equals()** method under the covers

Strings Are Immutable

TEKNOLOGISK INSTITUT

- Don't be fooled: All string operations return copies of strings!

```
string s1 = "Hello!";
s1.ToUpper();
Console.WriteLine( s1 == "Hello!" );          // ???
Console.WriteLine( s1 == "HELLO!" );          // ???

s1 += " Again...";
```

```
Console.WriteLine( s1[ 0 ] );
s1[ 0 ] = 'Y';
```

- **`System.Text.StringBuilder`** is specially designed for gradually building strings

TEKNOLOGISK
INSTITUT

Agenda

- .NET Common Type System
- Predefined Value Types
- Expressions
- Data Type Conversions
- User-defined Value Types
- Arrays
- Strings
- **Value Types Revisited – `Nullable`**

What Are Nullable Types?

TEKNOLOGISK INSTITUT

- Can assume the values of the underlying value type as well as **`null`**

```
int? i = 87;
int? j = null;
if( i.HasValue )
{
   int k = i.Value + j.GetValueOrDefault( 42 );
   Console.WriteLine( k );
}
```

```
int k = i.Value + ( j ?? 42 );
```

- The **`??`** operator is an elegant shorthand

TEKNOLOGISK INSTITUT

Characteristics of Nullable Types

- Make no mistake about it: Nullable types are **value types**!
- Only value types can be nullable!

- `int?` is actually defined as

```
Nullable<int> i = 42;
```

- This will become apparent when we discuss Generics later…

TEKNOLOGISK
INSTITUT

Summary

- .NET Common Type System
- Predefined Value Types
- Expressions
- Data Type Conversions
- User-defined Value Types
- Arrays
- Strings
- Value Types Revisited – `Nullable`

TEKNOLOGISK
INSTITUT

Question

- What will be written to the Console when executing the following statements?

```
Console.WriteLine( 87L / 2 == 87 / 2m );
```

a) Compile-time error
b) Runtime error
c) `True`
d) `False`

TEKNOLOGISK
INSTITUT