Module 4

"Reference Types and Statements"

TEKNOLOGISK
INSTITUT

TEKNOLOGISK INSTITUT

Agenda

- **Arrays**
- Strings
- Value Types Revisited – `Nullable`
- Selection Statements
- Iteration Statements
- Jump Statements
- Lab 4
- Discussion and Review

What Are Arrays?

- 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!

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;
```

Indexing Arrays

- Arrays are indexed by variable name and index

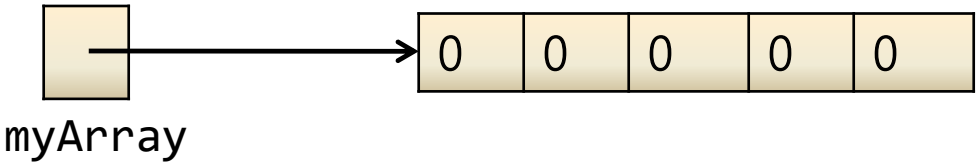
```
int[] myArray;
...
Console.WriteLine( myArray[2] ); // 112
```

42	87	112	...	256

Creating Arrays

- 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 ];
```



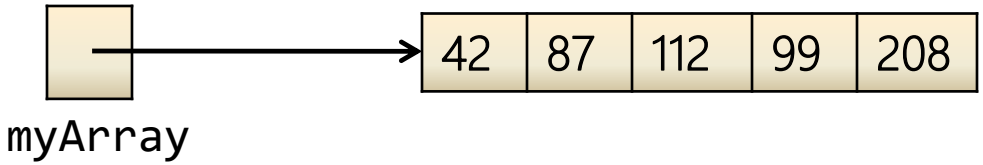

```
int[] myArray = new int[ 5 ];
```

- Arrays are by default initialized with “Zero Whitewash”

Initializing Arrays

- Arrays can be explicitly initialized

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



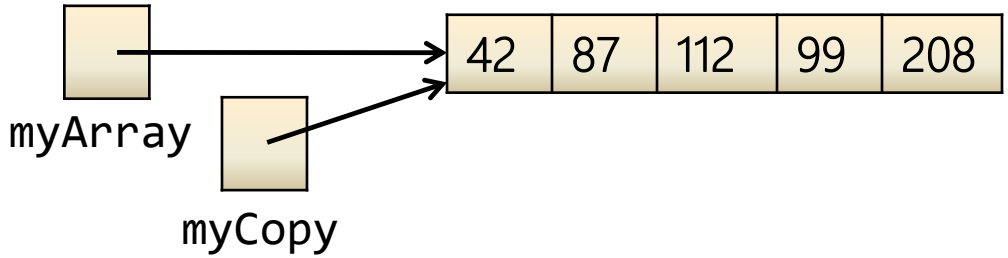
- A convenient shorter syntax exists

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

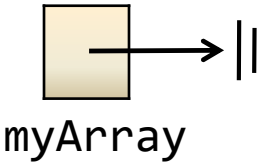
Assigning Array Variables

- 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;
```

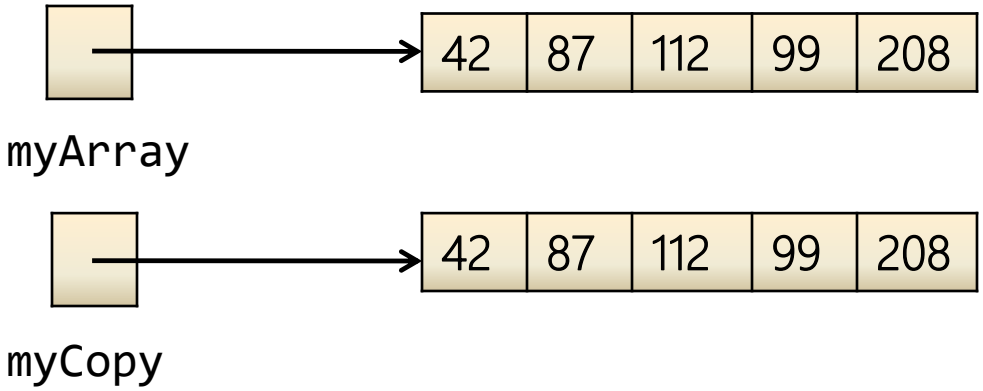


- This is the case for reference types in general

Comparing Array Variables

- 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

System.Array

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

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

TEKNOLOGISK INSTITUT

Agenda

- Arrays
- **Strings**
- Value Types Revisited – `Nullable`
- Selection Statements
- Iteration Statements
- Jump Statements
- Lab 4
- Discussion and Review

System.String

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

Interpolating Strings

- C# 6.0 offers easier formatting of strings
  - *String Interpolation* is essentially a built-in **`string.Format()`**

```
string firstName = "Bruce";
string lastName = "Campbell";

string name = $"{firstName} {lastName}";
```

- Very flexible
  - Works with any expression
  - Respects usual formatting characters etc.

Manipulating Strings

- 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

- 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

- 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

- Arrays
- Strings
- **Value Types Revisited – `Nullable`**
- Selection Statements
- Iteration Statements
- Jump Statements
- Lab 4
- Discussion and Review

What Are Nullable Types?

- 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

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…

Null-conditional Operator

- C# 6.0 introduces a new null-conditional operator **?.**
  - Also known as the "Elvis operator" ☺

```
string s = null;

// ...

string t = s?.ToUpper(); // Upper-case (or null if s == null)
```

- Right-hand side only evaluated if left-hand side is not null
  - Propagates null through expression

- Interacts brilliantly with the null-coalescing operator **??**

```
string t = s?.ToUpper() ?? "No string";
```

TEKNOLOGISK INSTITUT

Agenda

- Arrays
- Strings
- Value Types Revisited – `Nullable`
- **Selection Statements**
- Iteration Statements
- Jump Statements
- Lab 4
- Discussion and Review

`if-else` Statements

```
if( i > 0 )
{
   Console.WriteLine( "i is greater than 0" );
}
else
{
   Console.WriteLine( "i is 0 or less" );
}
```

- Condition must be Boolean
- Parenthesis are required!

- Use braces!

- `else`-branch is optional

Nested `if-else`

```
if( i > 100 )
{
   Console.WriteLine( "i is really large" );
}
else if( i > 10 )
{
   Console.WriteLine( "i is okay big" );
}
else if( i > 0 )
{
   Console.WriteLine( "i is big" );
}
else
{
   Console.WriteLine( "i is not much" );
}
```

- No **`elseif`** keyword

switch

- Switch handles a predefined set of choices

```
Console.WriteLine("1 [C#], 2 [VB]");
string langChoice = Console.ReadLine();
int n = int.Parse( langChoice );
switch( n )
{
   case 1:
      Console.WriteLine("Good choice, C# is the best!");
      break;
   case 2:
      Console.WriteLine("VB .NET: OOP and more!");
      break;
   default:
      Console.WriteLine("Well...good luck with that!");
      break;
}
```

TEKNOLOGISK INSTITUT

Agenda

- Arrays
- Strings
- Value Types Revisited – `Nullable`
- Selection Statements
- **Iteration Statements**
- Jump Statements
- Lab 4
- Discussion and Review

for Loop

- Uses Initialization, a terminating Condition, and an Incrementation statement

```
// Note! "i" is only visible within the for loop.
for( int i = 0; i < 4; i++ )
{
   Console.WriteLine( "Number is: {0} ", i );
}

// "i" is not visible here.
```

- Can iterate over several variables
- Any of the three can be left out

```
for( int i = 0, j = 10; ; i++, j -= 10 )
{
   Console.WriteLine( "i = {0}. j = {1}", i, j );
}
```

foreach Loop

- Iterates over all elements of an enumerable set

```
int[] myArray = { 42, 87, 112, 99, 208 };
foreach( int i in myArray )
{
   Console.WriteLine("Number is: {0} ", i);
}

// "i" is not visible here.
```

- Counter variable is read-only!

- Type must implement the **`IEnumerable`** interface
  - Works for a number of predefined as well as user-defined types
  - See Module 10

while Loop

- Iterates zero or more times
- Iterating Boolean condition is evaluated before each iteration
- Executes statement block if condition is true

```
string userIsDone = "";
while( userIsDone.ToLower() != "yes" )
{
   Console.Write("Are you done? [yes] [no]: ");
   userIsDone = Console.ReadLine();
}
```

- Condition must be Boolean
- Parentheses are required – braces are not

do-while Loop

- Iterates one or more times
- Iterating Boolean condition is evaluated after each iteration
- Executes statement block if condition is true

```
string userIsDone = "";
do
{
  Console.Write("Are you done? [yes] [no]: ");
  userIsDone = Console.ReadLine();
} while( userIsDone.ToLower() != "yes" );
```

- Condition must be Boolean
- Parentheses are required

Agenda

- Arrays
- Strings
- Value Types Revisited – `Nullable`
- Selection Statements
- Iteration Statements
- **Jump Statements**
- Lab 4
- Discussion and Review

continue

- Used in loop constructs
- Skips remainder of iteration

```
foreach (int i in myArray)
{
   if( i != 87 )
   {
      continue;
   }
   Console.WriteLine( "Number is: {0} ", i );
}
```

break

- Used in loop constructs
- Skips remainder of iteration and exits loop

```
foreach (int i in myArray)
{
   if( i == 87 )
   {
      Console.WriteLine( i );
      break;
   }
}
```

- Also used in **`switch`** statements

goto

- Redirects flow of control to a labeled statement

```
if( i == 42 )
{
   goto Mol;
}
goto End;
Mol:
  Console.WriteLine( 42 );
  goto End;
End:
```

- Also used in **`switch`** statements
- Avoid using **`goto`** except…

Quiz: Statements – Right or Wrong?

```
if( i = 0 )
{
   ...
}
```

```
if( myString.Length )
{
   ...
}
```

```
if( i == 87 )
   i++;
```

```
while i == 87
{
   ...
}
```

```
foreach( var i in myGrid )
{
   ...
}
```

```
do
   i++
while( i < 87 );
```

Lab 4: Using Reference Types and Statements in Programs

Discussion and review

- Arrays
- Strings
- Value Types Revisited – `Nullable`
- Selection Statements
- Iteration Statements
- Jump Statements

TEKNOLOGISK
INSTITUT

WINCUBATE

Jesper Gulmann Henriksen
PhD, MCT, MCSD, MCPD

Phone : +45 22 12 36 31
Email : jgh@wincubate.net
WWW : http://www.wincubate.net

Hasselvangen 243
8355 Solbjerg
Denmark