TEKNOLOGISK
INSTITUT

Module 4

Reference Types and Statements

Reference types (heap)

TEKNOLOGISK INSTITUT

- Reference types goes on the heap with a reference on the stack

1. `Person a;`
2. `a = new Person() { Name = "Mikkel", Born = 2003 };`

1

Stack	Heap
a = null;	

2

Stack	Heap
a = [ref] →	{ Name = ”Mikkel”, Born = 2003 }

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

Reference types (heap)

TEKNOLOGISK INSTITUT

- **References** are copied (not values)

```
1   Person a;
    Person b;

2   a = new Person() { Name = "Mikkel", Born = 2003 };
    b = new Person() { Name = "Mathias", Born = 2006 };

3   a = b;  // Reference to the SAME object
```

1

Stack	Heap
a = null; b = null;	

Stack	Heap

a = [ref]
b = [ref]
	{
	Name = ”Mikkel”,
	Born = 2003
	}
	{
	Name = ”Mathias”,
	Born = 2006
	}

3

Stack	Heap

a = [ref]
b = [ref]
	{
	Name = ”Mathias”,
	Born = 2006
	}

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

TEKNOLOGISK
INSTITUT

What Are Arrays?

- An array is a **reference type**
- 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 with []
- Array size is not a part of the declaration!
- You can declare arrays of several dimensions

```
int[] myIntArray;
int[,] myIntArray2;
string[] myStringArray;
string[,] myStringArray2;
```

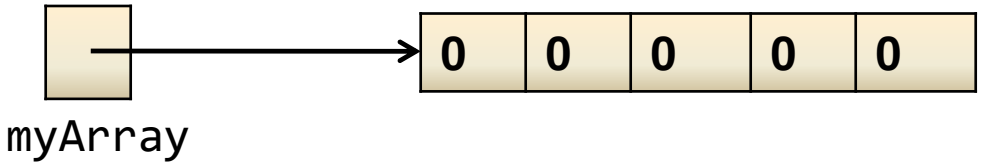
- This is only declaring an array by creating a variable that can reference an array in memory
  - It will not create the actual array in memory!!

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”

```
int[] myIntArray;
int[,] myIntArray2;
myIntArray = new int[10];
myIntArray2 = new int[10, 10];
string[] myStringArray = new string[10];
string[,] myStringArray2 = new string[10, 5];
```

TEKNOLOGISK
INSTITUT

Initializing Arrays

- Arrays can be explicitly initialized with {}

```
int[] myIntArray = { 5, 6, 3, 4, 6 };
string[] myStringArray = { "Mikkel", "Mathias" };
```

TEKNOLOGISK
INSTITUT

Indexing Arrays

- Arrays are indexed by variable name and index

```
int[] myIntArray = { 5, 6, 3, 4, 8 };
myIntArray[0] = 10;      // replace 5 with 10
int v = myIntArray[1];
Console.WriteLine(v);    // 6
```

TEKNOLOGISK INSTITUT

Assigning Array Variables

- Copying array variables amounts to copying references only!

```
int[] myArray = { 42, 87, 112, 99, 208 };
int[] myCopy;
myCopy = myArray; // reference is copied - not value
myArray[1] = 0;
int v = myCopy[1];
Console.WriteLine(v);   // 0
```

- Null value

```
int[] myArray = { 4, 5, 6 };
int v = myArray[0];
Console.WriteLine(v);   // 4
myArray = null;
Console.WriteLine(v);   // 4
v = myArray[0];         // Exception
```

- This is the case for reference types in general

TEKNOLOGISK
INSTITUT

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); // false
```

- This is the case for reference types in general

TEKNOLOGISK
INSTITUT

System.Array

- Arrays are instances of System.Array and that class has a lot of static methods
  - Clear()
  - Reverse()
  - Sort()
  - IndexOf()
  - Resize()

```
int[] myArray = { 4, 5, 2, 1, 8 };
System.Array.Sort(myArray);                 // myArray = 1,2,4,5,8
System.Array.Reverse(myArray);              // myArray = 8,5,4,2,1
int i = System.Array.IndexOf(myArray, 5);   // i = 1
System.Array.Resize(ref myArray, 6);        // myArray = 1,2,4,5,8,0
```

- The instance itself has methods and properties as well
  - Clone()
  - CopyTo()
  - Length

```
int[] myArray = { 4, 5, 2, 1, 8 };
int l = myArray.Length;                     // l = 5
int[] newArray = new int[5];
myArray.CopyTo(newArray, 0);                // newArray = 4,5,2,1,8
```

TEKNOLOGISK
INSTITUT

Array of arrays

- You can create arrays of arrays
  - Called “jagged arrays”
  - Visualize stack/heap ☺

```
// array of array
int[][] test = new int[2][];
test[0] = new int[10];
test[1] = new int[8];

test[0][0] = 5;
```

System.String

TEKNOLOGISK INSTITUT

- Represents text (Unicode)
- Is really a reference type but assignment, comparison, and manipulation are simplified
- String objects are immutable
  - Cannot be changed after they have been created
- Use ” when declaring a string constant
  - Use ‘ when declaring a single character (char - not a string)
- Strings have a number of useful instance methods and properties
  - Length, Compare(), Contains(), Format(), Insert(), PadLeft(), PadRight(), Remove(), Replace(), Split(), Substring(), Trim(), ToUpper(), ToLower()
- String Escape Sequences
  - \” (quote)
  - \\ (backslash)
  - \t (tab)
  - \f (formfeed)
  - \n (newline)
  - \r (carriage return)

```
string name1 = "Mikkel";
string name2 = null;
name2 = "Mathias";
if (name1 == name2) {    // not ref.compare

}
string name3 = name1 + " " + name2;    //Mikkel Cronberg
string f = name2.ToUpper();            //MIKKEL MATHIAS
```

TEKNOLOGISK
INSTITUT

Strings Are Immutable

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

```
string s1 = "Hello!";
s1.ToUpper();
Console.WriteLine(s1 == "Hello!");          // true
Console.WriteLine(s1 == "HELLO!");          // false
string s2 = s1.ToUpper();
Console.WriteLine(s2 == "HELLO!");          // true
```

- System.Text.StringBuilder is specially designed for gradually building strings
  - Don’t use System.String in loops

`if-else` Statements

TEKNOLOGISK
INSTITUT

```
int i = 2;
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

TEKNOLOGISK
INSTITUT

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");
}
```

TEKNOLOGISK
INSTITUT

switch

- Switch handles a predefined set of choices

```
int n = 1;
switch (n)
{
    case 1:
        Console.WriteLine("1...");
        break;
    case 2:
        Console.WriteLine("2...");
        break;
    default:
        Console.WriteLine("?...");
        break;
}
```

for Loop

TEKNOLOGISK INSTITUT

- 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 actually loop several variables, and contain multible conditions but it’s rarely used

TEKNOLOGISK
INSTITUT

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

TEKNOLOGISK
INSTITUT

`while` Loop

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

```
DateTime d2 = DateTime.Now.AddSeconds(2);
while (DateTime.Now < d2)
{
    Console.WriteLine("*");
}
```

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

TEKNOLOGISK
INSTITUT

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

TEKNOLOGISK
INSTITUT

continue

- Used in loop constructs
- Skips remainder of iteration

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

TEKNOLOGISK
INSTITUT

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