

Module 5

"Methods"



TEKNOLOGISK
INSTITUT

Agenda

- ▶ **Creating and Calling Methods**
- ▶ Passing Parameters
- ▶ Variations of Methods
- ▶ Lab 5
- ▶ Discussion and Review

The Syntax of a Method

- ▶ The syntax of methods are

```
ReturnValue MethodName( arguments ) { MethodBody }
```

- ▶ All methods must exist inside of a class definition – no “global” methods!

```
class Program
{
    static void DoStuff( )
    {
        Console.WriteLine( 87 );
    }
}
```

```
class Calculator
{
    public int Add(int x, int y)
    {
        return x + y;
    }
}
```

- ▶ `Main()` is a method that you already know
- ▶ `WriteLine()` is a method on the `Console` class

Local Variables

- ▶ Methods can declare local variables
 - Created during method invocation
 - Local to the method (i.e. "private")
 - Exist only inside method and are destroyed on exit
- ▶ Classes can declare member variables
 - These exist for the lifetime of the class
 - Can be used for sharing data
- ▶ Local variables take precedence over member variables!



Invoking a Method

- ▶ You can invoke a method **M** within the same class

```
M( 87, true );
```

- ▶ You can invoke a method **M** within another class **C**

```
C.M( 87, true );
```

- ▶ **M** must be visible to the outside, i.e. "public"
- ▶ You can invoke methods, which in turns invokes other methods etc. etc.
- ▶ Call Stack Window in Visual Studio



Returning from a Method

- ▶ The method returns
 - When the method body has finished executing
 - When a **return** statement is executed

```
static void DoStuff( )  
{  
    Console.WriteLine( 87 );  
}
```

```
static void DoMore( )  
{  
    int i;  
    ...  
    if( i < 0 ) { return; }  
  
    Console.WriteLine( 87 );  
}
```

Returning Values from Methods

- ▶ Methods can return values if declared with a specific return type (i.e. not **void**)

```
int cv;  
cv = CoolValue();  
Console.WriteLine( cv );
```

```
static int CoolValue( )  
{  
    int mol = 42;  
    ...  
    return mol + 87 - 112;  
}
```

- ▶ Values are returned with a **return** statement
- ▶ Must return a value of the specified return type!
- ▶ Return value is copied back
- ▶ Return value does not have to be used, however...

Implicit Typing in Methods

- ▶ The **var** keyword cannot be used as parameters or return value in methods

```
public var M( var x, var y )  
{  
    ...  
}
```

- ▶ But can be used locally inside the method body

```
int GetSomeInt()  
{  
    var ret = 87; ✓  
    return ret;  
}
```


Agenda

- ▶ Creating and Calling Methods
- ▶ **Passing Parameters**
- ▶ Variations of Methods
- ▶ Lab 5
- ▶ Discussion and Review

Passing Parameters by Value

- ▶ Define *formal* parameters within parentheses in method
 - Supply type and name for each parameter

```
static void Twice( int x )  
{  
    x = 2 * x;  
}
```

- ▶ Invoke method by supplying *actual* parameters in parentheses
 - The formal and actual parameter types and count must be compatible

```
int i = 42;  
Twice( i );  
Console.WriteLine( i );
```

- ▶ Parameter values are copied from actual to formal
- ▶ Changes made inside method has no effect outside method!



The ref Modifier

- ▶ Reference parameters are references to memory locations, i.e. aliases for variables
- ▶ Use the **ref** modifier to pass variables by reference

```
static void Twice( ref int x )  
{  
    x = 2 * x;  
}
```

```
int i = 42;  
Twice( ref i );  
Console.WriteLine( i );
```

- ▶ Also use the **ref** keyword when invoking the method
- ▶ Parameter values are referred (or aliased)
- ▶ Changes made inside method has indeed effect outside method!
- ▶ Variable must be assigned before call



The out Modifier

- ▶ Passing by reference consists of both “inputting” and “outputting”
- ▶ Use the **out** modifier when only outputting value

```
static void FillWithNumber( out int x )  
{  
    x = 87;  
}
```

```
int i;  
FillWithNumber( out i );  
Console.WriteLine( i );
```

- ▶ Also use the **out** keyword when invoking the method
- ▶ Parameter values are output
- ▶ Changes made inside method has indeed effect outside method!
- ▶ Variable does not have to be assigned before call



The params Modifier

- ▶ Passing parameter lists of varying length by using the **params** modifier

```
static int Sum( params int[] values )  
{  
    int total = 0;  
    foreach( int i in values )  
    {  
        total += i;  
    }  
    return total;  
}
```

```
Console.WriteLine( Sum( 42, 87 ) );
```

- ▶ Actual parameters are then passed into the method by value as an array
- ▶ Only one **params** per method



Optional Parameters

- ▶ Methods can have optional parameters by specifying their default values

```
static void M( int x, int y = 87, bool z = false )  
{  
    ...  
}
```

M(1, 2, true);	✓
M(1, 2); // Equivalent to M(1, 2, false)	✓
M(1); // Equivalent to M(1, 87, false)	✓
M(); // Illegal! x is required!	✗

- ▶ Optional parameters can be omitted when invoking the method
- ▶ Note: Optional parameters must appear last in parameter list
- ▶ Default values for optional parameters must be known at compile time!

```
static void N( bool b, DateTime dt = DateTime.Now )  
{  
    ...  
}
```



Named Parameters

- ▶ Can pass parameter values using their *names* (as opposed to their *position*)

```
M(1, z: true); ✓ // z is passed by name  
M(x: 1, z: true); ✓ // x and z are both passed by name  
M(z: true, x: 1); ✓ // z and x are both passed by name (equivalent!)  
M(z: true, 1 ) ✗
```

- ▶ Note: Positional parameters must always appear before any named parameters when invoking methods!
- ▶ Named and optional parameters mix perfectly
- ▶ Syntax look horrible, but what is the alternative...? ☺



Agenda

- ▶ Creating and Calling Methods
- ▶ Passing Parameters
- ▶ **Variations of Methods**
- ▶ Lab 5
- ▶ Discussion and Review

Overloading Methods

- ▶ Methods can be overloaded
 - Same name for multiple methods within a class

```
static int Add( int x, int y )  
{  
    return x + y;  
}  
static int Add( int x, int y, int z )  
{  
    return x + y + z;  
}  
static double Add( double a, double b )  
{  
    return a + b;  
}
```

```
Console.WriteLine( Add( 42, 87 ) );  
Console.WriteLine ( Add( 42, 87, 112 ) );  
Console.WriteLine( Add( 9.7, 0.1 ) );
```

- ▶ Compiler chooses correct method to invoke



Method Signatures

- ▶ Compiler chooses method based upon matching signatures
- ▶ *Signature* of a method consists of
 - Name
 - Parameter's type
 - Parameter's modifier (*)
- ▶ Note that certain things do not affect the signature
 - Parameter's name
 - Return type

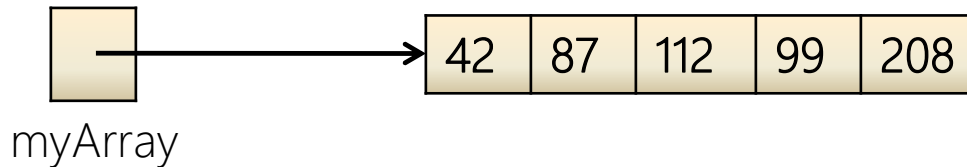


Methods and Reference Types

- Reference types can of course be passed to methods as well

```
static void Increment( int[] array )  
{  
    for( int i = 0 ; i < array.Length ; i++ )  
    {  
        array[ i ]++;  
    }  
}
```

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



- What do you think happens here?



Recursive Methods

- ▶ Methods can call itself either directly or indirectly.
- ▶ Such methods are said to be *recursive*

```
static int Factorial( int n )  
{  
    if( n <= 1 )  
    {  
        return 1;  
    }  
  
    return n * Factorial( n - 1 );  
}
```

```
Console.WriteLine( Factorial( 10 ) );
```

- ▶ Perfect for solving inductively defined problems
- ▶ Must have terminating base clause
- ▶ Use with care!



Quiz: Methods – Right or Wrong?

```
void M1( int x ) { ... }
```

```
M1( true );
```



```
void M2( string s ) { ... }
```

```
M2( "87" );
```



```
int M3( int x ) { x = x + 87; }
```

```
int y = M3( 42 );
```



```
int M4( int x ) { if( x < 99 ) { return x; } }
```

```
int y = M4( 42 );
```



```
void M5( out int x ) { x = 112; }
```

```
int y;  
M5( y );
```



```
int M6( int x = 42 ) { return x + 6; }
```

```
int y = M6();
```



```
void M7( int x = 0, int y = 1, int z = 2 ) { ... }
```

```
M7( , , 87 );
```



Lab 5: Creating Methods



Discussion and Review

- ▶ Creating and Calling Methods
- ▶ Passing Parameters
- ▶ Variations of Methods



WINCUBATE

Jesper Gulmann Henriksen

PhD, MCT, MCSD, MCPD

Phone : +45 22 12 36 31

Email : jgh@wincubate.net

WWW : <http://www.wincubate.net>

Hasselvangel 243

8355 Solbjerg

Denmark