Module 03

"Statements and Methods"

TEKNOLOGISK
INSTITUT

TEKNOLOGISK
INSTITUT

Agenda

- Selection Statements
- Iteration Statements
- Jump Statements
- Creating and Calling Methods
- Passing Parameters
- Variations of Methods

`if-else` Statements

TEKNOLOGISK INSTITUT

```
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`

TEKNOLOGISK INSTITUT

```
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

TEKNOLOGISK INSTITUT

- 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 a fine language.");
      break;
   case 2:
      Console.WriteLine("VB .NET: OOP, multithreading, and
more!");
      break;
   default:
      Console.WriteLine("Well...good luck with that!");
      break;
}
```

TEKNOLOGISK
INSTITUT

Agenda

- Selection Statements
- **Iteration Statements**
- Jump Statements
- Creating and Calling Methods
- Passing Parameters
- Variations of Methods

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 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 );
}
```

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 (more about that later)
  - Works for a number of predefined as well as user-defined types

while Loop

TEKNOLOGISK
INSTITUT

- 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

TEKNOLOGISK INSTITUT

- 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
- Braces are required

TEKNOLOGISK
INSTITUT

Agenda

- Selection Statements
- Iteration Statements
- **Jump Statements**
- Creating and Calling Methods
- Passing Parameters
- Variations of Methods

continue

TEKNOLOGISK INSTITUT

- 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

TEKNOLOGISK
INSTITUT

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…

TEKNOLOGISK
INSTITUT

Agenda

- Selection Statements
- Iteration Statements
- Jump Statements
- **Creating and Calling Methods**
- Passing Parameters
- Variations of Methods

The Syntax of a Method

TEKNOLOGISK INSTITUT

- 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

TEKNOLOGISK
INSTITUT

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!

TEKNOLOGISK INSTITUT

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 2012

TEKNOLOGISK
INSTITUT

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 );
}
```

TEKNOLOGISK INSTITUT

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

TEKNOLOGISK INSTITUT

- 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;
}
```

TEKNOLOGISK
INSTITUT

Agenda

- Selection Statements
- Iteration Statements
- Jump Statements
- Creating and Calling Methods
- **Passing Parameters**
- Variations of Methods

TEKNOLOGISK INSTITUT

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

TEKNOLOGISK INSTITUT

- 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 Modifer

TEKNOLOGISK INSTITUT

- 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

TEKNOLOGISK INSTITUT

- 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

TEKNOLOGISK INSTITUT

- 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

TEKNOLOGISK INSTITUT

- 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…? ☺

TEKNOLOGISK
INSTITUT

Agenda

- Selection Statements
- Iteration Statements
- Jump Statements
- Creating and Calling Methods
- Passing Parameters
- **Variations of Methods**

Overloading Methods

TEKNOLOGISK INSTITUT

- 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

TEKNOLOGISK INSTITUT

- 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

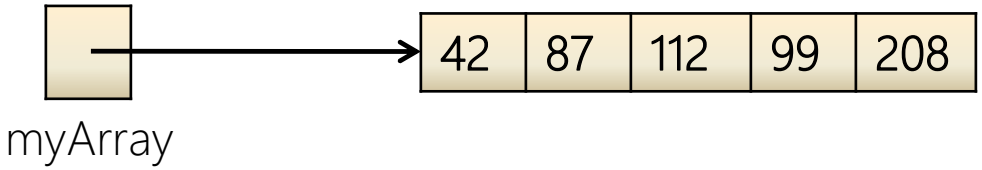
Methods and Reference Types

TEKNOLOGISK INSTITUT

- 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

TEKNOLOGISK INSTITUT

- 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!

TEKNOLOGISK
INSTITUT

Summary

- Selection Statements
- Iteration Statements
- Jump Statements
- Creating and Calling Methods
- Passing Parameters
- Variations of Methods

TEKNOLOGISK INSTITUT

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 );
```

Question

TEKNOLOGISK INSTITUT

You have written this program. Which code segment should you insert in the box for the program to keep asking for integer inputs until a valid integer is entered?

```
bool ok = false;
int i = 0;
do
{
   Console.WriteLine( "Input i:");
   ok = GetNumber( ref i );
} while ( !ok );
Console.WriteLine(
   "You entered: " + i );
```

```
static bool GetNumber( ref int number )
{
   string s = Console.ReadLine();
   int n;
   [                                    ]
   {
      number = n;
      return true;
   }
   return false;
}
```

a) `if( !int.TryParse( s, out n ) )`

b) `if( int.TryParse( s, out n ) )`

c) `if(( n=int32.Parse( s )) > int.MaxValue)`

d) `if(( n=int.Parse( s )) == double.NaN)`

TEKNOLOGISK
INSTITUT