

# Variables

Basic types (also defined as classes with methods):

Integer-types

Int, long, byte, sbyte, short, ushort, uint, ulong

Real-types

float, double , decimal (fixed decimal)

Bool-types

boolean: true / false

Char-types

char: ascii and unicode (16 bit)

Object types:

Always references (pointer) to object, allocated dynamic

Special:

String-types

string: object with lots of methods, constant value, operator-overload, acts nearly like basic types. Concatenation with + operator

# Calculation on numeric types

## Integer-types

- '+' : add fx  $x = y+z$ , where z is set to value of  $y+z$
  - '-' : subtract fx  $x = y-z$ , where z is set to value of  $y+z$
  - '\*' : multiply fx  $x = y*z$ , where z is set to value of  $y*z$
  - '/' : int divide fx  $x = y/z$ , where z is set to value of kvotient of  $y/z$
  - '%' : modulus divide fx  $x = y\%z$ , where x is set to value of rest of  $y/z$
- Normal precedens-rulls are uses including paranteses ()

## Real-types

As for the integer-types except the folowing

- '/' : real divide fx  $x = y/z$ , where z is set to value of  $y/z$  with decimals

# Calculation on numeric variables

## Short form of calculation

### Unary operators

'++': increment fx `++x` or `x++`, where z is incrementet by 1

'--': decrement fx `--x` or `x-`, where z is decrementet by 1

### Binary operators

'+=': add fx `x += y`, where x is set to value of `x+y`

'-=': subtract fx `x -= y`, where x is set to value of `x-y`

'\*=': multiply fx `x *= y`, where x is set to value of `x*y`

'/=': int divide fx `x /= y`, where x is set to value of kvotient of `x/y`

'%=': modulus divide fx `x %= y`, where x is set to value of rest of `x/y`

# Comparation on numeric values – result is boolean

'x == y': return true when x is equal to y, else false

'x != y': return true when x is not equal to y, else false

'x < y': return true when x is less than y, else false

'x <= y': return true when x is less than or equal y, else false

'x > y': return true when x is greater than y, else false

'x >= y': return true when x is greater than or equal y, else false

# Calculation on boolean values

## Unary operators

'!x': not x, returns true if `x==false`, otherwise false

## Binary operators

'x || y': x or y, returns true if x or y is true, else false

'x && y': x and y, return false if x and y is false, else true

Normal precedens-rulls are uses including paranteses ()

Example:

# Variables declaration

Variables can be declared as

- class variable ~ one instance per class
- object variable ~ one instance per object of the class
- parameter variable ~ one instance per start of method
- local variable (declared inside a scope in a method) ~ one instance per declaration.

Variables can be

- simple types (int, float,....., boolean and char)
- objects (references to)
- for parameters also special ref and out (references to variable), comes later in the course

# Excamples of variable declarations

```
class MyClass
{
    static int classvar; // a class variabel is declared as static
    int objectvar; // a object (instance) variable is declared without static keyword
    public void MyMethod (int parametervar) // a normal value parameter declaration
    {
        int localvar; // a normal local variabel declaration

        for (int forvar=1; forvar < 10; ++forvar) // a local variablel declaration lives in for-scope
        {
            .....
        }
        {
            int localscopevar // a variabel declaration for a local scope
            .....
        }
    }
}
```

# Livetimes of variables

## Class variable

- Lives from the first reference to the class to the end of application

## Object variable

- Lives from the object i created with new and until there is no referece to the object.
- Garbage-collection is done automatic.  
After the last reference to the object is gone, it is up to the garbage-collector to free the allocated memory.

*Continued*

# Livetimes of variables *(continued)*

## Parameter-variable (value parameter)

- Lives from the beginning of the method until it returns

## Local variables

- Lives from it is declared to end of the scope. The scope is starting with "{" and end with "}"
- Declaration in a for-construction lives inside the loop

Objects may live longer if other references to it or shorter if it is set to null.

# Reference and names of variables in an object method

```
class MyClass
{
    static int x , y, z;
    int u, v, w;
    public void MyObjectMethod (int x, int u, int a) // allowed
    {
        // not allowed int x,u,a; while name already declared as parameter
        int y, v; // allowed local variables - same name as in class and object
        int l; // allowed - new name
        l = x; // x is the parameter variable
        l = y; // y is the local variable
        l = z; // z is the class variable
        l = w; // w is the object (class instanse) variable
        l = MyClass.x; // x is the class variable
        l = this.u; // u is the object (class instanse) variable
    }
}
```

# Reference and names of variables in an class method

```
class MyClass
{
    static int x , y, z;
    int u, v, w;
    public static void MyClassMethod (int x, int u, int a) // allowed
    {
        // not allowed int x,u,a; while name already declared as parameter
        int y, v; // allowed local variables - same name as in class and object
        int l; // allowed - new name
        l = x; // x is the parameter variable
        l = y; // y is the local variable
        l = z; // z is the class variable
        l = w; // w is the object (class instanse) variable
        l = MyClass.x; // x is the class variable
        // not allowed l = this.u; // u is the object (class instanse) variable
    }
}
```

# Encapsulation of object and class variables

Variables of a class or object may be

- public: can be accessed direct outside the class / object
- internal: can be accessed direct outside the class / object but only from within the namespace
- protected: can be accessed direct from a inherited class but else not
- private: can not be accessed outside the class, neither from inherited class
- readonly: an extra keyword to the above.

A readonly variable can only be set when it is defined or in the constructors (like java final except may be set in constructors).

# Converting of numeric variables

Typecasting from real to integer (y is a float, double..)

```
int x = (int) float;  
long x = (long) float;
```

Converting from real and integer to string (x and y is float, int ....)

```
string s = x.ToString ()  
string s = x.ToString ( formating-string )  
- Fx x.ToString("F2") and y.ToString("N5")  
string s = string.Format ( formating-string, values ) ~ C's sprintf  
- Fx string.Format ( "x is {0:F2} and y is {1:N5}", x, y )  
Console.WriteLine(( formating-string, values ) ~ C's printf  
- Fx Console.WriteLine( "x is {0:F2} and y is {1:N5}", x, y )
```

Converting from string to real and integer (s is a string with the number)

```
int x = int.Parse ( s );  
float y = float.Parse ( s );
```

# Formatting codes (ordinary used)

C, c currency

D, d decimal

E, e Scientific exponential

F, f fixed-point

G, g general

N, n number

R, r roundtrip (format is to be converted back)

X, x hexadecimal

0 position-allocation for leading zero

# position-allocation for significant number

. decimal point char

, group separator

% percent char

; section separator

# Common escape characters for printing

- \' Single quote
- \\" Double quote
- \\" Backslash
- \0 Null
- \a Allert
- \b Backspace
- \f Formfeed
- \n Newline
- \r Carriage return
- \t Horizontal tab
- \v Vertical tab