Lexikalische Symbole

Namen

Syntax

```
Name = (letter | '_' | '@') {letter | digit | '_'}.
```

- Unicode!
- Groß/Kleinschreibung ist signifikant
- "@" am Anfang dient zur Unterscheidung von Namen und Schlüsselwörtern
  - if ... Schlüsselwort
  - @if ... Name *if*
- Können Unicode-Escapesequenz enthalten (z.B. \u03c0 für π)

Beispiele

someName	
sum_of3	
_10percent	
@while	Der Name *while*
π	Der Name π
\u03c0	Der Name π
b\u0061ck	Der Name *back*

Schlüsselwörter

abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	do	double	else	enum
event	explicit	extern	false	finally
fixed	float	for	foreach	goto
if	implicit	in	int	interface
internal	is	lock	long	namespace
new	null	object	operator	out
override	params	private	protected	public
readonly	ref	return	sbyte	sealed
short	sizeof	stackalloc	static	string
struct	switch	this	throw	true
try	typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual	void
while				

76 Schlüsselwörter in C# im Gegensatz zu 47 Schlüsselwörtern in Java

Namenskonventionen

Indirekt aus Base Class Library ableitbar

Groß/Kleinschreibung

- Jeden Wortanfang groß schreiben (z.B. *ShowDialog*)
- Anfangsbuchstabe groß, außer bei Variablen, Konstanten und Feldern, die man nicht von außen sieht.

Konstanten	klein	*size* (public-Konstanten groß, z.B. *MaxValue*)
Variablen	klein	*i*, *top*, *sum*
Felder	klein	*width*, *bufferLength* (public-Felder groß)
Properties	groß	*Length*, *FullName*
Enum-Konstanten	groß	*Red*, *Blue*
Methoden	groß	*Add*, *IndexOf*
Typen	groß	*StringBuilder* (vordefinierte Typen klein: int, string)
Namespaces	groß	*System*, *Collections*

Erstes Wort

- void-Methoden sollten mit Verb beginnen (z.B. *GetHashCode*)
- Alles andere sollte mit Substantiv beginnen (z.B. *size*, *IndexOf*, *Collections*)
- enum-Konstanten oder bool-Members können mit Adjektiv beginnen (*Red*, *Empty*)

Ganze Zahlen

Syntax

```
DecConstant = digit {digit} {IntSuffix}.
HexConstant = "0x" hexDigit {hexDigit} {IntSuffix}.
IntSuffix = 'u' | 'U' | 'l' | 'L'.
```

Typ

ohne Suffix:	kleinster aus int, uint, long, ulong
Suffix u, U:	kleinster aus uint, ulong
Suffix l, L:	kleinster aus long, ulong

Beispiele

17	int
9876543210	long
17L	long
17u	uint
0x3f	int
0x10000	long
0x3fL	long

Gleitkommazahlen

Syntax (vereinfacht)

```
RealConstant  = [Digits] ["." [Digits]] [Exp] [RealSuffix].
muß zumindest 1 Ziffer und etweder ".", Exp oder RealSuffix enthalten
Digits        = digit {digit}.
Exp           = ("e" | "E") ["+" | "-"] Digits.
RealSuffix    = "f" | "F" | "d" | "D" | "m" | "M".
```

Typ

ohne Suffix:	double
Suffix f, F:	float
Suffix d, D:	double
Suffix m, M:	decimal

Beispiele

3.14	double
1E-2	double
.1	double
10f	float

Zeichen und Zeichenketten

Syntax

```
CharConstant = ' char '.
StringConstant = " {char} ".
```

char kann sein

beliebiges Zeichen außer Ende-Hochkomma, Zeilenende oder \

Escape-Sequenz

\'	'
\"	"
\\	\
\0	0x0000
\a	0x0007 (alert)
\b	0x0008 (backspace)
\f	0x000c (form feed)
\n	0x000a (new line)
\r	0x000d (carriage return)
\t	0x0009 (horizontal tab)
\v	0x000b (vertical tab)

Unicode- oder Hex-Excape-Sequenz

\u0061	a
\x0061	a

Zeichen und Zeichenketten (Forts.)

Beispiele für Escape-Sequenzen in Zeichenketten

```
"file \"C:\\sample.txt\""                    file "C:\sample.txt"
"file \x0022C:\u005csample.txt\x0022"
```

Wenn @ vor einer Zeichenkette steht

- gilt \ nicht als Metazeichen
- wird " durch Verdopplung ausgedrückt
- dürfen Zeilenumbrüche vorkommen

Beispiel

```
@"file                                       file
""C:\sample.txt"""                           "C:\sample.txt"
```

Kommentare

Zeilenende-Kommentare

```
// a comment
```

Klammerkommentare

```
/* a comment */
```

Dürfen nicht geschachtelt werden

Dokumentationskommentare

```
/// a documentation comment
```