GK Informatik: Nand2Tetris	Jack Überblick
Name:	Datum:

Kommentar

//einzeiliger Kommentar Kommentar */* mehrzeiliger Kommentar */*

Keywords

Programm Komponenten:	*class, constructor, method, function*
Einfache Typen:	*int, boolean, char, void*
Deklaration von Variablen:	*var, static, field*
Statements:	*let, do, if, else, while, return*
Konstanten:	*true, false, null*
Referenz:	*this*

Symbole

( ): Listen von Parametern und Argumenten

[ ]: Indexierung von Arrays

{ }: Zusammenfassen von Programmteilen (Methoden, if-, else-, while-Statements)

; Ende von Statments , Separator von Variablen in Listen

= Vergleich und Zuweisung . Class-Operator

+, -, *, /, &, |, ~, <, >

Datentypen

```
var char c;  var String s;
let char c = 65;                   //c = ‘A’    Achtung!!! let char c = ‘A’; funktioniert nicht
let s = „A“;  let c = s.charAt(0); // funktioniert


var Array arr;                     // erzeugt Pointer Variable arr
let arr = 5000;                    // setzt die Basisadresse auf 5000
let arr[100] = 42                  // Speicher mit Adresse 5100 wird auf 42 gesetzt
```

Standard IO

```
let zahl = Keyboard.readInt(”Gib eine Zahl ein:  ”);
do Output.printString( ”Deine Zahl ist ” );
do Output.printInt( zahl );
```

Variables

Variable kind	Description	Declared in	Scope
static variables	`static` *type varName*1, *varName2*, … ; Only one copy of each static variable exists, and this copy is shared by all the object instances of the class (like *private static variables* in Java)	class declaration	The class in which they are declared.
field variables	`field` *type varName*1, *varName2*, … ; Every object (instance of the class) has a private copy of the field variables (like *member variables* in Java)	class declaration	The class in which they are declared, except for functions, where they are undefined.
local variables	`var` *type varName*1, *varName2*, … ; Local variables are created just before the subroutine starts running and are disposed when it returns (like *local variables* in Java)	subroutine declaration	The subroutine in which they are declared.
parameter variables	*type varName*1, *varName2*, … Used to pass arguments to the subroutine. Treated like local variables whose values are initialized "from the outside", just before the subroutine starts running.	subroutine signature	The subroutine in which they are declared.

Jack's standard class library / OS

OS class	***Services***
`Math`	Common mathematical operations: `multiply(int,int)`, `sqrt(int)`, etc.
`String`	Represents string objects and related methods: `length()`, `charAt(int)`, etc.
`Array`	Represents array objects and related operations: `new(int)`, `dispose()`.
`Output`	Supports text output to the screen: `printString(String)`, `printInt(int)`, `println()`, etc.
`Screen`	Supports graphics output to the screen: `drawPixel(int,int)`, `setColor(boolean)`, `drawCircle(int,int,int)`, etc.
`Keyboard`	Supports input from the keyboard: `readLine(String)`, `readInt(String)`, etc.
`Memory`	Facilitates access to the host RAM: `peek(int)`, `poke(int,int)`, `alloc(int)`, `deAlloc(Array)`.
`Sys`	Supports execution-related services: `halt()`, `wait(int)`, etc.

Statements

Statement	Syntax	Description
`let`	`let` *varName* `=` *expression*`;` or `let` *varName*`[`*expression1*`] =` *expression2*`;`	An assignment operation (where *varName* is either single-valued or an array). The variable kind may be *static*, *local*, *field*, or *parameter*.
`if`	`if (`*expression*`) {` *statements1* `}` `else {` *statements2* `}`	Typical *if* statement with an optional *else* clause. The curly brackets are mandatory even if *statements* is a single statement.
`while`	`while (`*expression*`) {` *statements* `}`	Typical *while* statement. The curly brackets are mandatory even if *statements* is a single statement.
`do`	`do` *function-or-method-call*`;`	Used to call a function or a method for its effect, ignoring the returned value.
`return`	`Return` *expression*`;` or `return;`	Used to return a value from a subroutine. The second form must be used by functions and methods that return a void value. Constructors must return the expression `this`.

The Hack character set

key	code
(space)	32
!	33
"	34
#	35
$	36
%	37
&	38
'	39
(	40
)	41
*	42
+	43
,	44
-	45
.	46
/	47

key	code
0	48
1	49
…	…
9	57

:	58
;	59
<	60
=	61
>	62
?	63
@	64

key	code
A	65
B	66
C	…
…	…
Z	90

[	91
/	92
]	93
^	94
_	95
`	96

key	code
a	97
b	98
c	99
…	…
z	122

{	123
|	124
}	125
~	126

key	code
newline	128
backspace	129
left arrow	130
up arrow	131
right arrow	132
down arrow	133
home	134
end	135
Page up	136
Page down	137
insert	138
delete	139
esc	140
f1	141
…	…
f12	152

Compilieren und Programm starten

- Öffne *cmd* und gehe ins Verzeichnis \nand2tetris\tools.
- Compile Verzeichnis mit allen Jack Dateien, z.B.: *JackCompiler ..\\projects\\09\\Square*
- Öffne *VMEmulator!*
- Lade die internen OS-Bibliotheken!
- Lade das gesamte Verzeichnis (z.B. *Square*).
- Ggf. schalte die Animation aus: *Animate -> no animation*.