

ESOP - Simple Programs

Assoc. Prof. Dr. Mathias Lux
ITEC / AAU

Agenda



- Symbols
- Variables, Constants
- Assignments
- Operators

Symbols: names



Naming of variables, types, functions, etc. within a program.

- consist of letters, digits and ,_‘
- always start with a letter
- arbitrary length
- case sensitive
- Examples
 - x, x17, my_Var, myVar

Symbols: key words



- Name key parts of the program
- cannot be used as names
- Examples:
 - if, while, for, enum, class, static, ...

Symbols: numbers



- Discrete numbers
 - (decimal or hexadecimal)
- Floating point numbers
- Examples
 - 376 ... decimal
 - 0x1A5 ... hexadecimal
 - 3.14 ... floating point

Symbols: strings



- Any strings between quotation marks.
- Must not exceed end of lines
- " needs to be escaped to \"
- Examples
 - "a simple string"
 - "she said \"Hallo\""

Symbols: strings



- String
 - in Java not a base data type but an object.
- char ... single Unicode letter
 - 2 Bytes
 - simple apostrophe, eg. 'L', ')', ...

Declaration of variables



- Each variable must be declared before use
 - Name and type are given to the compiler
 - Compiler allocates memory
- Examples:
 - `int x; ...` declares variable `x` of type `int` (integer)
 - `short a, b; ...` declares two variables of type `short` (short integer)

Integer types



byte	8 bit	$-2^7 \dots 2^7-1$	(-128 .. 127)
short	16 bit	$-2^{15} \dots 2^{15}-1$	(-32.768 .. 32.767)
int	32 bit	$-2^{31} \dots 2^{31}-1$	(-2.147.483.648 ..)
long	64 bit

- Declaration & initialisation

- `int x = 100;`
declares integer x and assign value of 100.
- `short a = 0, b = 1;`
declares two short variables with initial values.

Constants



- Init variables that cannot be changed later
 - `static final` `int max = 100;`
- Why would you do that?
 - readability
 - max easier to read than 100
 - maintainability
 - if the same value is used several times.
- Constants are declared in class scope
 - will be explained later in the course

Comments



- line comments
 - Start with `//` .. and with end-of-line (EOL)
- block comments
 - use `/* ... */`, can span over multiple lines.

- Comments & Readability
 - comment for later understanding
 - do not comment what's obvious

```
// Hier ist ein Zeilenkommentar

int x = 15; // Initialisierung an dieser Stelle erforderlich!
short y = -12;

/* *****
   Dieses Programm wurde von Mathias Lux geschrieben
   ***** */
```

Language for comments and names







- Think about your audience
 - English is better than German
- Do not mix languages!
- Special care with
 - swear words, email addresses, people names, licenses!

Search

shit

Search

 Repositories	203
 Code	26,340
 Issues	2,815
 Users	31

Languages

C	224,353
HTML	34,242
JavaScript	x
GAS	25,739
Less	23,338
C++	18,093
PHP	15,306
XML	9,977
Ruby	9,436
Markdown	8,795

[Advanced search](#) [Cheat sheet](#)

We've found 26,340 code results

Sort: Best match ▾

 [romankalb/PMapp – main.js](#) JavaScript
Last indexed on 3 Aug

```
1 debug("Shit");
```

 [matthewwcv/nodestuff – mmcolors.js](#) JavaScript
Last indexed on 31 Jul

```
1 alert('shit');
```

 [lwl8851206/HelloWorld – test.js](#) JavaScript
Last indexed on 29 Jul

```
1 function shit(){};
```

 [ACSysFMI/theDoctors – shit.js](#) JavaScript
Last indexed on 25 Jul

```
1 alert('shit');
```

 [nitirajrathore/testrepo – tits.js](#) JavaScript
Last indexed on 2 Aug

```
1 alert("fucking dick shit");
```

 [gpestana/legacy_slick.js – core_tests.js](#) JavaScript
Last indexed on 1 Aug

```
1 /*Tests and shit.. */
```

 [bmelon11/myrepo – boo.js](#) JavaScript
Last indexed on 28 Jul

```
1 console.log("eat shit");
```

 [apiengine/apiengine-client – page.js](#) JavaScript
Last indexed on 23 Jul

```
1 some profile shit goes here
```

 [AchintyaAshok/NYT---Intern-Project-Front-End – storyView.js](#) JavaScript
Last indexed on 28 Jul

```
1 console.log('django is shit');
```

 [JamieAppleseed/jamieappleseed.com – application.js](#) JavaScript
Last indexed on 8 Aug

```
1 (function(win){
2   // do shit
3 })(this);
```



Choice of variables and constants



- Coding conventions exist for
 - readability of code
 - maintainability and preservation
- Naming conventions see:
<http://www.oracle.com/technetwork/java/javase/documentation/codeconventions-135099.html#367>
- Tipps:
 - Names that make sense (cp. comments)
 - Better shorter than longer (cp. support by IDE).

No good naming ..



[aeonsf/footlocker – dpc-hashcrack.py](#)

Last indexed on 31 Jul

Python

```
51         return licker
52
53     def toptobottom(crack):
54         i = 0
55         while i < (len(asshole)/2):
56             if len(crack) == 32:
57                 if crack == md5(asshole[i]):
58
59
60                 if crack == md5(asshole[i]):
61                     print "\n\t[p1] 3===D passwd is = %s\n"%asshole[i]
62                     break
63             elif len(crack) == 40:
```



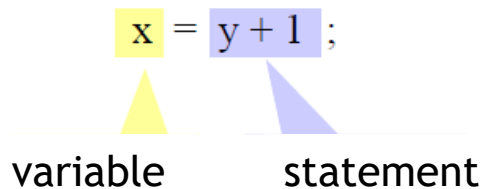
[hallfox/teampython – ex10b.py](#)

Last indexed on 3 Aug

Python

```
5     escape4 = "%s is a total asshole."
6
7     asshole = "Tyler \t\nFUCK\n Sontag \\"
8
9     singlequoteformatting = '''
10    This looks a lot cleaner and minimalistic.
11
12    For now on, let's use the single quotes instead.
13    '''
14
15    print escape1
16    print escape2
17    print escape3
18    print escape4 % asshole
19    print singlequoteformatting
```

Assignments



1. compute statement
2. store in variable

- left and right side have to be compatible
 - either the same type (int, byte, ...)
 - or type left \supseteq type right
- hierarchy of integer types
 - long \supseteq int \supseteq short \supseteq byte

Assignments



- Examples

```
int i, j; short s; byte b;
```

```
i = j;           // ok: same type
```

```
i = 300;         // ok (numeric expressions are int)
```

```
b = 300;         // not ok: 300 > byte
```

```
i = s;          // ok
```

```
s = i;          // not ok
```

Static Type Check



- Compiler checks:
 - variables stay in allowed value range.
 - operators are applied on the right types / values.

Arithmetic Expressions



- Simplified grammar

`Expr = Operand {BinaryOperator Operand}.`

`Operand = [UnaryOperator] (identifier | number | "(" Expr ")").`

- eg. `- x + 3 * (y + 1)`

Arithmetic Expressions



- Binary Operators

+ sum

– subtraction

* multiplikation

/ division $5/3 = 1$

% modulo $5\%3 = 2$

- Unary operators

+ identity $(+x) = x$

– invert sign

Types in Arithmetic Expressions



- Order of operations
 - multiplication and division (*, /, %) over addition and subtraction (+, -)
 - eg. $2 + 3 * 4 = 14$
 - left association
 - eg. $7 - 3 - 2 = 2$
 - unary operators over binary operators
 - eg.: $-2 * 4 + 3$ ergibt -5
- Resulting types
 - input type can be byte, short, int, long
 - resulting type
 - if one operand is long -> result is type long,
 - otherwise -> type int

Examples

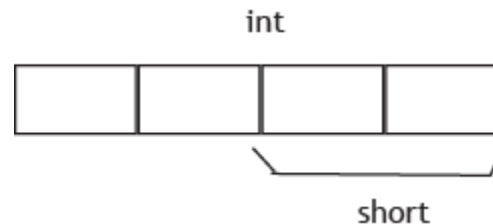


```
short s; int i; long x;  
x = x + i;           // long  
i = s + 1;           // int (1 is int)  
s = s + 1;           // false!  
s = (short)(s + 1);  // type cast necessary
```

Type Cast

(type) expression

- changes *expression* to *type*
- result can be truncated



Increment / Decrement



- access variable plus operation
 - `x++` ... returns `x` and then adds `+1`
 - `++x` ... adds `1` to `x` and then returns `x`
 - `x--` , `--x` ... the same with subtraction.
- can be a statement on its own right
 - `x = 1; x++;` // `x = 2` the same as: `x = x + 1;`
- examples
 - `x = 1; y = x++ * 3;` // `x = 2, y = 3` is: `y = x * 3; x = x + 1;`
 - `x = 1; y = ++x * 3;` // `x = 2, y = 6` is: `x = x + 1; y = x * 3;`
- only works on variables, not expressions.
 - `y = (x + 1)++;` // wrong!

The power of two (shifts)



Shift-operators allow for efficient multiplication and division by powers of two.

Multiplication

$x * 2$	$x \ll 1$
$x * 4$	$x \ll 2$
$x * 8$	$x \ll 3$
$x * 16$	$x \ll 4$
...	...

Division

$x / 2$	$x \gg 1$
$x / 4$	$x \gg 2$
$x / 8$	$x \gg 3$
$x / 16$	$x \gg 4$
...	...

Division only works out for positive numbers.

The power of two (shifts)



Examples

`x = 3;`

0000 0011

`x = x << 2; // 12`

0000 1100

`x = -3;`

1111 1101

`x = x << 1; // -6`

1111 1010

`x = 15;`

0000 1111

`x = x >> 2; // 3`

0000 0011

Assignment operators.



- arithmetic operations can be combined with assignments.

	short	long
<code>+=</code>	<code>x += y;</code>	<code>x = x + y;</code>
<code>-=</code>	<code>x -= y;</code>	<code>x = x - y;</code>
<code>*=</code>	<code>x *= y;</code>	<code>x = x * y;</code>
<code>/=</code>	<code>x /= y;</code>	<code>x = x / y;</code>
<code>%=</code>	<code>x %= y;</code>	<code>x = x % y;</code>

String Operators



- Strings can be concatenated with ‘+’
 - “Mathias” + “ ” + “Lux”
- Other operators do not apply
 - Especially not comparisons
 - “Mathias” != “Lux” ... checks addresses!

Bit Operators



- Bits of operands are modified
 - Example(Java uses two's complement)
 - byte a = 17; // 00010001
 - byte b = 7; // 00000111
- Supported operations
 - Disjunction:
 - byte or = a | b; // 23
 - Conjunction:
 - byte and = a & b; // 1
 - Antivalence:
 - byte xor = a ^ b; // 22
 - Complement:
 - byte notB = ~b; // -8

Java-Programs



```
class ProgramName {  
    public static void main (String[] arg) {  
        ... // Declarations  
        ... // Statements  
    }  
}
```

Text has to be in file named
ProgramName.java

// Example:

```
class Sample {  
    public static void main (String[] arg) {  
        int a = 23;  
        int b = 100;  
        System.out.print("Sum = ");  
        System.out.println(a + b);  
    }  
}
```

```
C:\Windows\system32\cmd.exe  
E:\Temp>javac Sample.java  
E:\Temp>java Sample  
Summe = 123  
E:\Temp>
```

Compile and Run with JDK



- Compile

- C:\> cd MySamples
change to source file
- C:\MySamples> javac Sample.java
create class file (compile)

- Execute

- C:\MySamples> java Sample
run class file
- Sum = 123

Example: IDEA IDE



- Strings, comments and variables
 - Spell check, consistency, type check
- Live Templates
 - psvm + <tab>
- Automated naming of Variables
 - <Strg>-<Space>