MODULE 1-05: Compact Course Programming

Lesson 4

- Data Types and Operators -

Prof. Dr. Rolf Schuster
Computer Science Department
University of Applied Science and Arts Dortmund
Rolf.Schuster@fh-dortmund.de

Content

- **Simple Data Types, their Values and Operators**
- Expressions
- Type Conversions

Try out / Answer: Overflow and Precision

DO: Try out the following calculations in BlueJ code pad!

- Overflow: What results do you get for „mystery“

```
int oneMillion = 1000000;
int mystery = oneMillion * oneMillion;
```

- Precision: What results do you get for „total price“

```
double unitPrice = 4.35;
double totalPrice = 100 * unitPrice;
```

Go to the following link to check your answer:
Udacity Link: https://classroom.udacity.com/courses/cs046/lessons/192345866/concepts/1923908140923#

Simple Data Types in Java

Fachhochschule Dortmund
University of Applied Sciences and Arts

Type Name	Content	Value Range	Default-Value	Size
`boolean`	Locic Value	`true`, `false`	`false`	1 Bytes
`char`	Unicode-Character	`\u0000` … `\uffff` (0 … 65535)	`\u0000`	2 Bytes
`byte`	Whole Number with +/- Sign	-128 … 127	`0`	1 Bytes
`short`	Whole Number with +/- Sign	-32768 … 32767	`0`	2 Bytes
`int`	Whole Number with +/- Sign	-2147483648 … 2147483647	`0`	4 Bytes
`long`	Whole Number with +/- Sign	-9223372036854775808 … 9223372036854775807	`0`	8 Bytes
`float`	Floating Point Number	+/- $3.4028235 \cdot 10^{38}$ appr. 7 significant decimal places	`0.0f`	4 Bytes
`double`	Floating Point Number	+/-$1.7976931348623157 \cdot 10^{308}$ appr. 7 significant decimal places	`0.0d`	8 Bytes

Simple Data Types in Java

In difference to some other programming languages, all simple data types in Java have an agreed **fixed size** in memory

For every simple data type a **default value** is defined, which is of importance with the initialisation of object and class variables (note: local variables are not automatically initialised with the default value)

Operators

Are special symbols that are used to link operands to determine a new value

According to their number of operands we can distinguish three types of operators

- **Single digit** (**monadic** oder **unary**) operators
  example: the negative sign −
- **Two digit** (**dyadic** oder **binary**) operators
  example: the addition sign `+`
- **Three digit** (**triadic** oder **ternary**) operatores
  example: conditional operator `? :`

Logic Values (`boolean`)

Unary Operator (Operator Operand)

Operator	Description
`!`	logische Negation

Binary Operator (Operand1 Operator Operand2)

Operator	Description	Example
`==`	Equality	`false == true` results in `false`
`!=`	Inequality	`false != true` results in `true`
`&`	logic AND	
`|`	logic OR	
`^`	logic XOR	

Logic Value (`boolean`)

Notation

Area	NEGATION	AND	OR	Exklusiv-OR
Mathematics / Logic	¬a	a ∧ b	a ∨ b	a ⊕ b
Java	!a	a & b	a | b	a ^ b

Properties of the Operators (truth table)

a	!a
false	true
true	false

a	b	a & b	a | b	a ^ b
false	false	false	false	false
false	true	false	true	true
true	false	false	true	true
true	true	true	true	false

Truth Table (`boolean`)

Boolean Expressions with **&** and **|** evaluate both terms completely

In practise complete evaluation is often not required

- **&** - operation: is one expression `false`, then the overall result is `false`
- **|** - operation: is one expression `true`, then the overall result is `true`

The operators **&&** and **||** ensure a shortened evaluation

Example

```
int m = 3;
int n = 5;
boolean b = (m < 5) || (n > 3);
```

`n>3` is not evaluated!

- The shortened evaluation is important to reduce the computation time for example with large arrays

Try out / Answer: use boolean operators!

What is the value of the following expressions?

1. `(true & true) | false`
2. `!!true`
3. `true & !true`
4. `(true || false) && true`

Binary Operators (`char`, `byte`, `short`, `int`, `long`)

Operator	Description	Example
+	Addition	5 + 6 yields 11
-	Subtraction	9 - 3 yields 6
*	Multiplication	10 * 15 yields 150
/	Whole Number Division	13 / 3 yields 4
%	Modulo (remainder of a whole number division)	20 % 7 yields 6
<	smaller	3 < 5 yields true
<=	smaller equal	3 <= 3 yields true
>	bigger	2 > 10 yields false
>=	bigger equal	5 >= 6 yields false
==	equal	3 == 3 yields true
!=	not equal	5 != 5 yields false

Try out / Answer: use boolean operators!

What is the value of the following expressions?

```
1.   7 / 5
2.   7 % 5
3.   5 / 7
4.   5 % 7
```

Unary Operators (`char`, `byte`, `short`, `int`, `long`)

Operator	Description	Example
`-`	Unary Negation	`-i`
`++`	Increment	`++i` is the same as `i = i+1`
`--`	Decrement	`--i` is the same as `i = i-1`

Note the difference: **Pre-**increment vs. **Post-**increment

```
a = ++b;     // is the same as:
                // b = b+1; a = b;
a = b++;     // is the same as:
                // a = b; b = b+1;
```

Bitwise - Operators (`char`, `byte`, `short`, `int`, `long`)

Access to binary representation of whole number data types

Numbers are viewed as a set of consecutive bits, which may be manipulated

Unary Operator (Operator Operand)

Operator	Description
~	Complement (bitwise negation)

Binary Operators (Operand1 Operator Operand2)

Operator	Description
&	bitwise AND
|	bitwise OR
^	bitwise XOR

- The operators >>, >>> and << are used to shift the bits to the right or the left

Bitwise - Operators (`char`, `byte`, `short`, `int`, `long`)

Example for shift operator

– Left-Shift-Operator <<

```
int a;
a = 10;
a << 3;
```

lost bits

000**00000** 00000000 00000000 00001010

00000000 00000000 00000000 01010**000**

filled with bits

```
int a;
a = -10;
a << 3;
```

lost bits

11111111 11111111 11111111 11110110

11111111 11111111 11111111 10110**000**

filled with bits

Equivalent to: whole-number multiplication with 2^3

Floating-Point-Numbers (`float`, `double`)

Unary Operators (analog to whole-number types)

`-` `++` `--`

Binary Operators (analog to whole-number types)

`+` `-` `*` `/` `%` (arithmetic operators)

`<` `<=` `>` `>=` `==` `!=` (comparison operators)

Note:

- whole-number division: 45 / 20 Result: 2
- floating-point division: 45.0 / 20.0 Result: 2.25

Floating-Point-Numbers (`float`, `double`)

Arithmetic Operators

- Both operands of type `float`
  - Result type `float`
- In all other cases
  - Result type `double`

Attention with equality checks

```
(x == y)    // possible rounding errors!
```

Composite Operators

E1 **op=** E2 is the same as E1 **=** E1 **op** (E2)

Example

```
counter = counter + 1;
// abbreviated: counter += 1;

counter = counter - 1;
// abbreviated: counter -= 1;
```

- analog: ***=**, **/=**, **%=**, **&=**, **|=**, **^=**, **<<=**, **>>=**, **>>>=**

Content

- Simple Data Types, their Values and Operators
- **Expressions**
- Type Conversions

Expressions: Definition and Features

Expression: Processing specification, that delivers a value after execution

- In the simplest case a variable or a constant
- Through combination of operands, operations and round brackets we get complex expressions

Examples

```
radius = 5.5;
area = PI * radius * radius;
counter = counter + 1;
```

Brackets

The evaluation of expressions in brackets always takes place first

- Just like the rules in mathematics

Expressions may be arbitrarily nested

Notation of the nested structure is done with round brackets

All expressions are provided in linear notation

- Expression are provided in line format

Example

Fachhochschule Dortmund
University of Applied Sciences and Arts

Prof. Rolf Schuster

Mathematic format	Line format
$\frac{k \cdot t \cdot p}{100 \cdot 360}$	`k * t * p / (100 * 360)`
$\frac{a \cdot f + c \cdot d}{a \cdot e - b \cdot d}$	`(a * f + c * d) / (a * e - b * d)`
$a + \frac{b}{d + \frac{e}{f + \frac{g}{h}}}$	`a + b / (d + e / (f + g / h))`
$B_0 \cdot \left(1 - n \cdot \frac{p}{100}\right)$	`B0 * (1 - n * p / 100)`

Operator Priority Rules

Well-known from mathematics:

- „Point before Line“
  - example 6 + 7 * 3 equals 27 and not 39

In Java:

- Linking of operators is governed by *priorities*:
  - An operator with high priority links stronger than an operator with a lower priority
  - If the priority is the same than then the *associativity* of the operators is evaluated
    - op is is *left associative*: X op Y op Z equals (X op Y) op Z
    - op ist *right associative*: X op Y op Z equals X op (Y op Z)
  - Obviously, brackets do control the evaluation order
    - example (6 + 7) * 3 ist 39

Priority and Associativity

Priority	Operators	Description	Associativity
14	`[]`	Field(Array)index	L
	`()`	Method call	L
	`.`	Component access	L
13	`++ --`	Pre- oder post-increment or –decrement	R
	`+ -`	Sign (unary)	R
	`~`	Bitwise complement	R
	`!`	Logic negation	R
	`(type)`	Type conversion	R
	`new`	Object generation	R
12	`* / %`	Multiplication, division, modulo	L
11	`- +`	Subtraction, addition and string-chaining	L
10	`<<`	Left shift	L
	`>>`	Right shift with sign	L
	`>>>`	Right shift without sign	L

Priority and Associativity

Priority	Operators	Description	Associativity
9	`< <= > >=`	Comparison: smaller, smaller or equal, bigger, bigger or equal	L
	`instanceof`	Type check of object	L
8	`== !=`	Equal, not equal	L
7	`&`	Logic-, bitwise AND	L
6	`^`	Logic-, bitwise Exclusiv-OR	L
5	`|`	Logic-, bitwise OR	L
4	`&&`	Logic AND	L
3	`||`	Logic OR	L
2	`? :`	Conditioned evaluation	L
1	`=`	Assignment	R
	`*= /= %=`	Combined assignment	R
	`+= -= <<=`		R
	`>>= >>>=`		R
	`&= ^= |=`		R

Try out / Answer: use priority and associativity!

For the following expressions, set brackets such that they yield the same result as the expressions without brackets

1. `e = --c - d / a;`
2. `f = b <= a || c > 16;`
3. `h = a < 5 || b > 10 && d - c >= 0;`

Expressions
Mathematical Constants and Functions

The class `Math` provides important mathematical constants and functions (see online documentation)

Constants

- `public static final double E` (basis e of nat. logarithm)
- `public static final double PI` (π)
- Usage: `Math.E` and `Math.PI`

Methods (Selection)

- `public static double abs(double x)` $|x|$
- `public static double cos(double x)` $\cos(x)$
- `public static double sin(double x)` $\sin(x)$
- `public static double tan(double x)` $\tan(x)$
- `public static double sqrt(double x)` $\sqrt{x}$
- `public static double exp(double x)` e^x
- `public static double pow(double x, double y)` x^y
- Usage: for example `result = Math.pow(a,b);`

Definition of Constants

Constants in Java

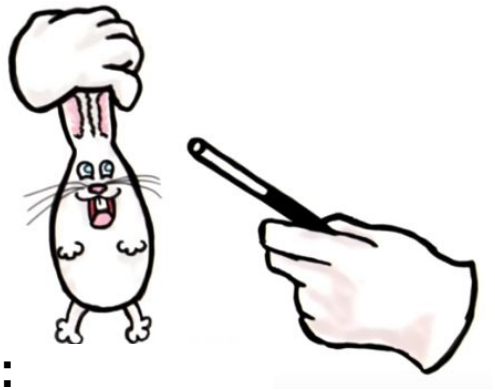
- Constants are defined and initialised like variables with the keyword „final“
- their names are typically written in capital letters
- they can not be changed

Example:

Statement and definition of constants:

```
red = Math.min(red + 25, 255);

final int ADDED_RED = 25;
final int MAX_RED = 255;
```



Statement rewritten with constants:

```
red = Math.min(red + ADDED_RED, MAX_RED)
```

Udacity Link: https://classroom.udacity.com/courses/cs046/lessons/192345866/concepts/1923908620923#

Content

- Simple Data Types, their Values and Operators
- Expressions
- **Type Conversions**

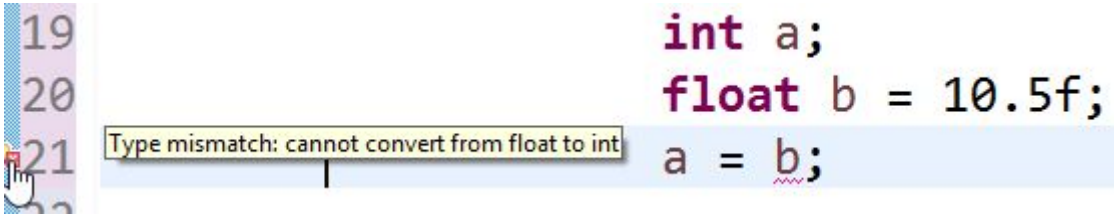
Type Conflict

Values can only be assigned to variables,

if their type is compatible with the type of the variable!

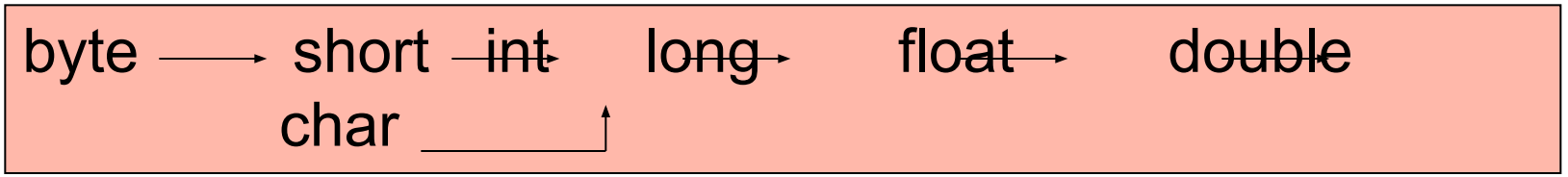
Example

```
int a;
float b = 10.5f;
a = b;  // Error because of incompatible types
```



Automatic (implicit) type extension

Rules



– An automatic type extension is happening in the direction of the arrows

Example

```
double a, b;
float c;
a = b + c + 2.785f;
```

Type extension and selection of operators in expressions

Each expression is evaluated step by step according to the priorities and associativity of its operators

The operators choosen are the operators that fit to the type of the operands (example: whole number division OR division for floating point numbers)

Are the types of operands different, than the „smaller“ operand will receive an automatic type conversion

Are both operands of an operation, expressions themselves, then the left operand is calculated before the right operand

Type Extension and Selection of Operators in Expressions

- Example

```
double a;
int b, c;
a = 3.0 + 2.785f + b / c;
```

- Evaluation: Addition from left to right, point before line

3) Addition of double- und int-value: int-value extended to double and + for double

```
a = ((3.0 + 2.785f)      +      (b / c));
```

1) 3.0 double, 2.785f float => type extension to double 2.785 and + for double-values

2) Both operands of type int => whole number division

Explicit Type Conversion: Type Casting

Explicit type conversion happens when the desired type is explicitly requested

Example

```
int a;
float b = 10.25f;
a = (int) b;
```

`float`-value `10.25f` is converted in the `int`-value 10

```
a = (int)(b / 3.3f + 5.73f);
```

Note: Type casting takes place AFTER the calculation of the entire term `b / 3.3f + 5.73f`

Reading Input from the Console

Example code to read an integer and a double from the keyboard:

- Remember to import the utilisation class „Scanner“

```
import java.util.Scanner;

public class InputDemo
{
  public static void main(String[] args)
  {
    Scanner in = new Scanner(System.in);
    System.out.print("How old are you? ");
    int age = in.nextInt();

    System.out.print("Next year, you will be ");
    System.out.println(age + 1);

    System.out.print("What is your weight? ");
    double weight = in.nextDouble();

    System.out.print("I hope next year that'll be ");
    System.out.print(weight * 0.95);
  }
}
```

- Imports the Scanner-class
- Reading an integer from the console
- Reading an double from the console

Udacity Link: https://classroom.udacity.com/courses/cs046/lessons/192345866/concepts/1923908660923

Formatted Output

Example code to print a number in a specific format:

```
public class FormatDemo
{
 public static void main(String[] args)
 {
   int quantity = 100;
   double unitPrice = 4.35;
   double totalPrice = quantity * unitPrice;

   System.out.print("Total: ");
   System.out.printf("%8.2f\n", totalPrice);

   double taxRate = 0.08;
   double tax = totalPrice * taxRate;

   System.out.print("Tax:   ");
   System.out.printf("%8.2f\n", tax);
 }
}
```

Printf-Formatting with argument „%8.2f\n“:

% - print something

8 - print total of 8 digits

.2 - with 2 digits after the decimal point

f - floating point number

\n - print a new line

Udacity Link: https://classroom.udacity.com/courses/cs046/lessons/192345866/concepts/1923908690923#

Try out / Answer: Overflow and Precision

DO: Fill in the empty fields!

```
int cookiesPerDay;
double cerealBoxesPerDay;
String name;
System.out.printf("%  ", cookiesPerDay);
System.out.printf(  , cerealBoxesPerDay);
System.out.printf(  , name);
```

what wide? goes here if I want to print 6 characters

what format string goes here if I want 2 decimal places 4 characters wide?

what format string goes here

Refer to the fact sheet for further details: https://www.udacity.com/wiki/cs046/factsheets

Go to the following link to check your answer:

Udacity Link: https://classroom.udacity.com/courses/cs046/lessons/192345866/concepts/1923908700923#

Homework Assignment 04 (2 Bonus Points)

(Assignment submission date provided in „Ilias … Homework Assignments“

Write the program “Milage Printer” in BlueJ…

- …that asks the user to input the following values
  - The number of gallons currently in the tank
  - The fuel efficiency in miles per gallon
- and then prints how far the car can go on the ga in the tank and the cost of driving 100 miles.
- Print the distance with 1 decimal point and the cost with 2 decimals
- Use System.out.print and not System.out.println Otherwise your output will not be formatted correctly
- Assume the cost per gallon is $3.95. Define it as a constant: final double COST_PER_GALLON = 3.95;
- If value entered for efficiency is less than or equal to 0, print "No can go". Otherwise continue with the calculations.
- Your output should be in the exact format shown below. The text will be identical - only the numbers will change.
- Important: Be sure to print the strings exactly as shown

Sample runs for the final version:

Enter the number of gallons of gas in the tank
5.1
Enter the fuel efficiency 35.0
Distance: 178.5
Cost: 11.29

Or:

Enter the number of gallons of gas in the tank
25
Enter the fuel efficiency -5
No can go

Go to the following link to try out your code: