Programmieren II

Operators and Variables & Types

Alexander Fraser

fraser@cl.uni-heidelberg.de

(Contains material from T. Bögel, K. Spreyer, S. Ponzetto, M. Hartung)

April 24, 2014

Organizational

- Klausur 23.07.2014?
- Tutoren: Felix Krauss 9-11 Tuesday, Julian Gerhard Thursday 18-20
- Moodle: ss14prog2

Outline

1. Recap
2. Data types
3. Operators
4. Variables

Outline

1 Recap

2 Data types

3 Operators

4 Variables

Recap

Java

- Platform independent
- Bytecode-compiled
- Secure![a]

[a] Not! `http://java-0day.com/`

Hello World

- Source files, classes, methods, statements
- Java development workflow

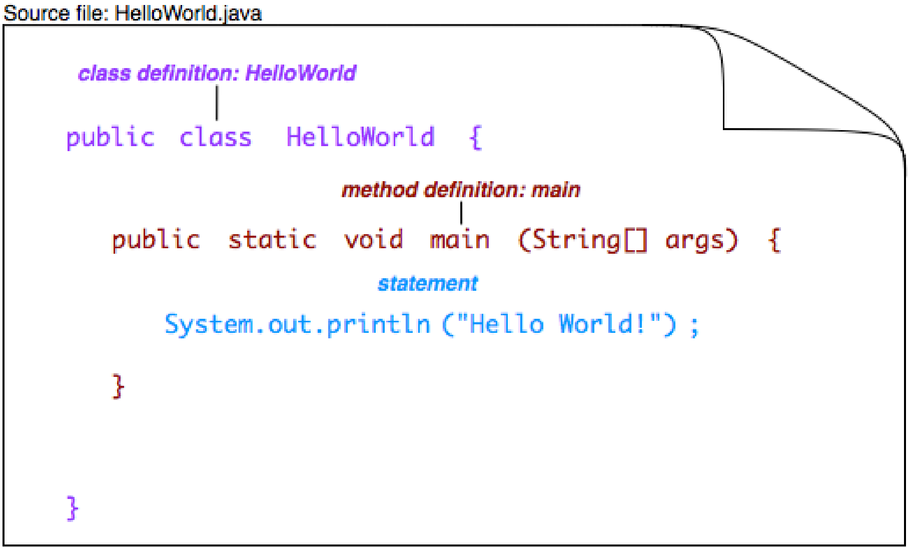
Hello World!

```java
// file name == class name
public class HelloWorld {
    // a method (function) called "main"
    // parameter input: array of strings
    public static void main( String[] args ) {
     System.out.println( "Hello world!" );
    }
}
```

code/HelloWorld.java

1. Save
2. Compile: `javac HelloWorld.java` → `HelloWorld.class`
3. Run: `java HelloWorld`

Anatomy of a class I



```
public class HelloWorld {

    public static void main (String[] args) {

        System.out.println ("Hello World!") ;

    }


}
```

Anatomy of a class II

Layered structure (simplified)

- source file (`.java`): *one class definition*
- class definition: one or more *methods*
- method: one or more *statements*

Goals of this session

- Understanding the concept of strong typing
- Choosing the correct data type for your implementation
- Learn about variables, references and assignments
- Learn how to perform arithmetic operations

Imperative programming

- *Core* of java: imperative programming (embedded into object oriented programming)
- Programs: step-by-step sequences of statements
- Statements represent values
- Values can be assigned to/represented by variables
- Tools for coordinating the flow of statements: *control structures*

Overview of basic syntactic components

Example: given a list L that contains n natural numbers, find the largest number in L.

```
int max = L[0];

for ( int i = 1; i < L.length; i++ ) {
    if ( L[i] > max ) {
       max = L[i];
    }
}
```

Overview of basic syntactic components

Example: given a list L that contains n natural numbers, find the largest number in L.

```
int max = L[0];

for ( int i = 1; i < L.length; i++ ) {
    if ( L[i] > max ) {
       max = L[i];
    }
}
```

$\Rightarrow$ Variables

Overview of basic syntactic components

Example: given a list L that contains n natural numbers, find the largest number in L.

```
int max = L[0];

for ( int i = 1; i < L.length; i++ ) {
    if ( L[i] > max ) {
        max = L[i];
    }
}
```

$\Rightarrow$ Type of a variable

Overview of basic syntactic components

Example: given a list L that contains n natural numbers, find the largest number in L.

```
int max = L[0];

for ( int i = 1; i < L.length; i++ ) {
    if ( L[i] > max ) {
       max = L[i];
    }
}
```

$\Rightarrow$ Assignments (i++ means i=i+1)

Overview of basic syntactic components

Example: given a list L that contains n natural numbers, find the largest number in L.

```
int max = L[0];

for ( int i = 1; i < L.length; i++ ) {
    if ( L[i] > max ) {
      max = L[i];
    }
}
```

$\Rightarrow$ Statements

Overview of basic syntactic components

Example: given a list L that contains n natural numbers, find the largest number in L.

```
int max = L[0];

for ( int i = 1; i < L.length; i++ ) {
    if ( L[i] > max ) {
       max = L[i];
    }
}
```

⇒ Statement, declaration of a variable

Overview of basic syntactic components

Example: given a list L that contains n natural numbers, find the largest number in L.

```
int max = L[0];

for ( int i = 1; i < L.length; i++ ) {
   if ( L[i] > max ) {
      max = L[i];
   }
}
```

⇒ Statements

Each statement ends with a semicolon (";")!

Overview of basic syntactic components

Example: given a list L that contains n natural numbers, find the largest number in L.

```
int max = L[0];

for ( int i = 1; i < L.length; i++ ) {
    if ( L[i] > max ) {
       max = L[i];
    }
}
```

⇒ Control structures (for loop, if condition)

- A word about style - per request, I added a recommended style guide to the slides from the first lecture (see the end)
- One thing recommended by most style guides: always use brackets, i.e., do not write:

  ```
  if (x) System.out.println(“true”);
  ```
- This will compile and work right, but it is hard to read

Outline

1 Recap

2 Data types

3 Operators

4 Variables

Variables, data types and values

- *Variables* denote positions in main memory, where values are stored
- Java uses *strong typing*: a variable that is instantiated with a certain type cannot change its type (implicitly) – in contrast to Python
- Each variable has a unique data type and can only take values of its type (static typing)

Example: static typing

```java
public class StaticTyping {
  public static void main(String[] args) {
    String a = "3";
    a = 4;
    System.out.println( a );
  }
}
```

code/StaticTyping.java

```java
public class StaticTyping2 {

  public static void main(String[] args) {
    int a = 3;
    System.out.println(a);
    a = "Ich bin ein String";
    System.out.println(a);
  }
}
```

code/StaticTyping2.java

Detour: main memory (1)

- Main memory (primary storage) of a computer consists of a sequence of bits without gaps. The content of each bit is either 0 or 1, depending on its electric state.
- Bits are aggregated to Bytes, which consist of 8 consecutive bits. A Byte is the smallest unit that can be accessed and manipulated in main memory.

Detour: main memory (2)

- Each Byte of the main memory is located at a unique and immutable position: *memory address*
- Memory addresses are enumerated from 0 in ascending order
- Each Byte in the main memory can take one of 256 possible values (values between 0 and 255): $2^8 = 256$

Sneak preview: Java Heap Space

`http://www.javacodegeeks.com/2012/09/java-memory-model-simplified.html`

- Part of memory allocated for a java program: heap space (expandable)
- Memory for new objects are allocated from the Heap
- Garbage collector operates on heap space
- Size of the heap space can be increased with JVM parameters (`-Xmx1G`)

Data types

- A data type represents a set of possible values and defines how a sequence of memory is interpreted
- Java distinguishes between two categories of types:
  - primitive data types
  - reference types
- Today: primitive types
- Classes (next week) are reference types

Primitive data types: whole numbers

- There are five data types for whole numbers that differ with respect to the range of possible values and memory requirement
- The following four types represent signed whole numbers:

`byte`	8 Bits
`short`	16 Bits
`int`	32 Bits
`long`	64 Bits

- Range of values: $-2^{b-1}, ..., 2^{b-1} - 1$ (b is the number of bits)
  e.g. `byte`: -128,...,127 (256 values, 2^8)

Primitive data types: whole numbers

- char uses 16 Bits and is unsigned
- represents values from 0 to 65,535
- interpretation as character in unicode encoding which covers almost all international alphabets
- Java fully supports unicode (in variable names, for instance)

Primitive data types: floating point numbers

- Two data types for floating point numbers:

`float`	32 Bits
`double`	64 Bits

- Values are encoded following IEEE standard (algebraic sign, exponent, fixed-point part)
- Range: $\sim \pm 10^{-45} .. \pm 10^{38}$ (float), $\pm 10^{-324} .. \pm 10^{308}$ (double).
- Beware of rounding errors!

Rounding errors with floating point numbers

The exactness of a floating point number is restricted (and depends on the amount of memory it occupies)

⇒ Beware of rounding errors!

Example

```
float x = 0.0644456f;
float y = 0.032754f;
float z = x * y;
System.out.println(z);
```

Output: 0.0021108512
Correct: 0.0021108511824

Primitive data types: truth values

- `boolean` represents truth values (`true` and `false`)

Memory allocation

- Problem: administer all students attend two courses:

  ```
  byte students_prog2;
  byte students_studpro;
  byte sum;
  sum = students_prog2 + students_studpro;
  ```

- What happens if 70 people attend “Programmieren II” and 65 people attend the programming project?

Memory allocation

- Problem: administer all students attend two courses:

  ```
  byte students_prog2;
  byte students_studpro;
  byte sum;
  sum = students_prog2 + students_studpro;
  ```

- What happens if 70 people attend “Programmieren II” and 65 people attend the programming project?

⇒ byte represents values up to +127 ⇒ throws an exception (compile-time or run-time)

Statements

- Values are represented by expressions
- Four different expressions
  - literals
  - variables
  - statements with operators
  - method calls

Literals

- Literals are expressions to represent a value
- Example: `2`, `3.14`, `true`, “`Hello`”
- Each literal encodes a unique data type
- Literals for whole numbers, floating point numbers, boolean values and strings

Literals for whole numbers

- Numbers (e.g. `17`, `0`, `-238723`) in the source code are regarded as literals of type `int` to the basis 10 per default
- Literals starting with 0 (except 0) are interpreted as octal numbers (basis 8): 013 represents 11
- Literals starting with 0x are interpreted as hexadecimal numbers (basis 16): 0x1cA represents 458
- Literals starting or ending with `L` oder `l` are interpreted as `long` values indicating that they represent large values

Literals for whole numbers: char

- Char literals are single characters in single quotes: 'a', 'Ä'.
- Special characters are marked with a \(backslash)
  - '\n' line break
  - '\'' single quote
  - '\\' backslash
- It's also possible to directly enter a unicode value: '\u1F494' represents 💔.

Literals for floating point numbers

- Literals like 3.14 und -2.6 containing a decimal point are interpreted as *double* values.
- Literals for floating point numbers can contain exponents (basis 10):
- 3.14e1 $\rightarrow$ 31.4, -2.e-2 $\rightarrow$ -0.02
- Suffix f stands for the type float; suffix d stands for double
- If a suffix or an exponent is used, the decimal point can be omitted: 0f, -2e-2 etc.

Other literals

- The key words `true` and `false` are literals of the data type `boolean`.

A small example

```java
public class Literals {

  public static void main(String[] args) {
    System.out.println(2);
    System.out.println(”Hallo”);
    System.out.println(-2e-2);
  }
}
```

Literals.java

A small example

```
public class Literals {

  public static void main(String[] args) {
    System.out.println(2);
    System.out.println(”Hallo”);
    System.out.println(-2e-2);
  }
}
```

Literals.java

```
% java Literals
```

A small example

```java
public class Literals {

  public static void main(String[] args) {
    System.out.println(2);
    System.out.println(”Hallo”);
    System.out.println(-2e-2);
  }
}
```

Literals.java

```
% java Literals
2
Hallo
-0.02
```

Outline

1 Recap

2 Data types

3 Operators

4 Variables

Operators

- Basic arithmetic operations are represented by operators:
  $2 + 3 \quad 0.4 * 25.4 - 100 \quad 27 > -3$
- Each operator expression has a unique data type.
- Different operators:
  - arithmetic operators
  - comparison operators
  - logic operators
  - various others

Arithmetic operators

- There is an arithmetic operator for basic operations (a, b refer to other expressions):

  a+b

  a-b

  a * b

  a/b

  a%b (Modulo)

- Arithmetic operators are used to combine numeric values of certain types and return corresponding numeric value types

Comparison operators

- Standard comparison operators:

$a<b$	$a>b$	
$a\leq b$	$a\geq b$	(greater than or equal)
$a == b$		(equal)
$a \mathrel{!=} b$		(unequal)

- For greater, greater than or equal etc., numeric values are used as operands.
- Equal and unequal can be used with arbitrary data types
- The result value of a comparison is always of type boolean

“==” vs. “=”

- “=” assigns a value to a variable
- “==” compares two values with each other
- Beware of compile time errors!

Example

```
int a = 5;
int b = 5;
if ( a == 5 ) {
    a++;
}
```

Widening type conversions

- What happens, if the operands of an operator have different types?
- Only numeric values: conversion to data type with *highest value range*
  - Whole numbers are converted into whole numbers with more bytes
  - `float` is converted to `double`
  - Whole numbers are converted to `float` or `double` (Beware of rounding errors!)

Logic operators

- Logic operators expect values of type boolean and return a value of the same type boolean:

a && b	“and”
a || b	“or”
!a	“not”

Example

```
if ( a==5 && b == 5 ) System.out.println(“both 5!”);
```

Operator precedence

- Operators can be nested: 2*3+4
- Bracketing to make precedence explicit: (2*3)+4 2*(3+4)
- Without brackets, Java performs multiplication and division first, then addition and subtraction

Operator precedence

Operatoren		associativity
() [] .		from left
++ −−	(prefix)	from right
++ −−	(postfix)	from right
!	(unary)	from right
* / %		from left
+ -		from left
<< >>	(bit-wise shift)	from left
< ≤ > ≥		from left
== !=		from left
&	(bit-wise AND)	from left
ˆ	(bit-wise XOR)	from left
|	(bit-wise OR)	from left
&&	(logic AND)	from left
||	(logic OR)	from left
?:	(conditional)	from right
= += -= *= /= %=		from right

Java calculator

```java
public class Calculator {
  public static void main(String[] args) {
    System.out.println(2+3*4);
    System.out.println(27*(7%2) > 30);
    System.out.println(103.02 - 2e-2);
    System.out.println(false || (3 > 2));
  }
}
```

Calculator.java

Java calculator

```java
public class Calculator {
  public static void main(String[] args) {
    System.out.println(2+3*4);
    System.out.println(27*(7%2) > 30);
    System.out.println(103.02 - 2e-2);
    System.out.println(false || (3 > 2));
  }
}
```

Calculator.java

```
% java Calculator
```

Java calculator

```java
public class Calculator {
  public static void main(String[] args) {
    System.out.println(2+3*4);
    System.out.println(27*(7%2) > 30);
    System.out.println(103.02 - 2e-2);
    System.out.println(false || (3 > 2));
  }
}
```

Calculator.java

```
% java Calculator
14
```

Java calculator

```java
public class Calculator {
  public static void main(String[] args) {
    System.out.println(2+3*4);
    System.out.println(27*(7%2) > 30);
    System.out.println(103.02 - 2e-2);
    System.out.println(false || (3 > 2));
  }
}
```

Calculator.java

```
% java Calculator
14
false
```

Java calculator

```java
public class Calculator {
  public static void main(String[] args) {
    System.out.println(2+3*4);
    System.out.println(27*(7%2) > 30);
    System.out.println(103.02 - 2e-2);
    System.out.println(false || (3 > 2));
  }
}
```

Calculator.java

```
% java Calculator
14
false
103.0
```

Java calculator

```java
public class Calculator {
  public static void main(String[] args) {
    System.out.println(2+3*4);
    System.out.println(27*(7%2) > 30);
    System.out.println(103.02 - 2e-2);
    System.out.println(false || (3 > 2));
  }
}
```

Calculator.java

```
% java Calculator
14
false
103.0
true
```

Outline

1 Recap

2 Data types

3 Operators

4 Variables

Variables

- Each variable contains the values of a specific data type
- Variables need to be declared (i.e. the compiler needs to know the data type)
- Values of a statement can be assigned to variables
- Variables can be evaluated to use their value in a statement

Example

```
public class Test {
   public static void main(String[] args) {
      int iq;
      iq = 2;
      iq = iq + 1;
      System.out.println(iq);
   }
}
```

Example

```
public class Test {
   public static void main(String[] args) {
      int iq;
      iq = 2;
      iq = iq + 1;
      System.out.println(iq);
   }
}
```

⇒ declaration as a variable of type `int`

Example

```
public class Test {
   public static void main(String[] args) {
      int iq;
      iq = 2;
      iq = iq + 1;
      System.out.println(iq);
   }
}
```

⇒ assignment of a value to a variable

Example

```
public class Test {
   public static void main(String[] args) {
      int iq;
      iq = 2;
      iq = iq + 1;
      System.out.println(iq);
   }
}
```

⇒ evaluation

Names for variables

- Variable names can consist of arbitrary unicode strings
- First character of a name: no digit
- Keywords (e.g. `for`, `if` etc.) can't be used as variable names
- Variable names are case-sensitive!
- Examples:
  x abc27 üß_27π

Declaration

```
typ var1 [= init1] [, var2 [= init2], ...];
```

- Multiple variables of the same type can be declared at once:
  `int a, b, c;`
- Variables can be initialized with values upon declaration:
  `int a = 2, _b100 = 27, my_var;`

Assignments

```
var = expr;
```

- The statement expr is evaluated and the value is assigned to var
- Widening type conversion possible
- Narrowing type conversion possible under certain conditions (usually: explicit cast!)

Assignment: Example 1

```java
public class WideningConversion {
  public static void main(String[] args) {
    float a = 0.5f;
    double b = a;
    System.out.println(b);
  }
}
```

WideningConversion.java

Assignment: Example 1

```java
public class WideningConversion {
  public static void main(String[] args) {
    float a = 0.5f;
    double b = a;
    System.out.println(b);
  }
}
```

WideningConversion.java

```
% javac WideningConversion.java
```

Assignment: Example 1

```java
public class WideningConversion {
  public static void main(String[] args) {
    float a = 0.5f;
    double b = a;
    System.out.println(b);
  }
}
```

WideningConversion.java

```
% javac WideningConversion.java
%
```

Assignment: Example 1

```java
public class WideningConversion {
  public static void main(String[] args) {
    float a = 0.5f;
    double b = a;
    System.out.println(b);
  }
}
```

WideningConversion.java

```
% javac WideningConversion.java
% java WideningConversion
0.5
```

Assignment: Example 1

```java
public class WideningConversion {
  public static void main(String[] args) {
    float a = 0.5f;
    double b = a;
    System.out.println(b);
  }
}
```

WideningConversion.java

```
% javac WideningConversion.java
% java WideningConversion
0.5
```

⇒ a is style of type `float` but b is assigned the value 0.5 as double, i.e. 0.5 is converted from `float` to `double`

Assignment: Example 2

```java
public class NarrowingConversion {
  public static void main(String[] args) {
    double a = 0.5d;
    float b = a;
    System.out.println(b);
  }
}
```

NarrowingConversion.java

Assignment: Example 2

```java
public class NarrowingConversion {
  public static void main(String[] args) {
    double a = 0.5d;
    float b = a;
    System.out.println(b);
  }
}
```

NarrowingConversion.java

```
% javac NarrowingConversion.java
```

Assignment: Example 2

```java
public class NarrowingConversion {
  public static void main(String[] args) {
    double a = 0.5d;
    float b = a;
    System.out.println(b);
  }
}
```

NarrowingConversion.java

```
% javac NarrowingConversion.java
NarrowingConversion.java:4: possible loss of precision
   found : double
   required: float
```

Assignment: Example 3

```java
public class NarrowingConversionImproved {
  public static void main(String[] args) {
    double a = 0.5d;
    float b = (float)a;
    System.out.println(b);
  }
}
```

NarrowingConversionImproved.java

Assignment: Example 3

```java
public class NarrowingConversionImproved {
  public static void main(String[] args) {
    double a = 0.5d;
    float b = (float)a;
    System.out.println(b);
  }
}
```

NarrowingConversionImproved.java

```
% javac NarrowingConversionImproved.java
```

Assignment: Example 3

```java
public class NarrowingConversionImproved {
  public static void main(String[] args) {
    double a = 0.5d;
    float b = (float)a;
    System.out.println(b);
  }
}
```

NarrowingConversionImproved.java

```
% javac NarrowingConversionImproved.java
%
```

Assignment: Example 3

```java
public class NarrowingConversionImproved {
  public static void main(String[] args) {
    double a = 0.5d;
    float b = (float)a;
    System.out.println(b);
  }
}
```

NarrowingConversionImproved.java

```
% javac NarrowingConversionImproved.java
% java NarrowingConversionImproved
```

Assignment: Example 3

```java
public class NarrowingConversionImproved {
  public static void main(String[] args) {
    double a = 0.5d;
    float b = (float)a;
    System.out.println(b);
  }
}
```

NarrowingConversionImproved.java

```
% javac NarrowingConversionImproved.java
% java NarrowingConversionImproved
0.5
```

Assignment: Example 3

```java
public class NarrowingConversionImproved {
  public static void main(String[] args) {
    double a = 0.5d;
    float b = (float)a;
    System.out.println(b);
  }
}
```

NarrowingConversionImproved.java

```
% javac NarrowingConversionImproved.java
% java NarrowingConversionImproved
0.5
```

⇒ use explicit casts when performing a narrowing conversion (not only to prevent compiler warnings)

Assignment: Example 4

What if you don't know what you're doing?

```java
public class NarrowingConversionEvil {
  public static void main(String[] args) {
    int a = 200;
    byte b = (byte)a;
    System.out.println(b);
  }
}
```

NarrowingConversionEvil.java

Assignment: Example 4

What if you don’t know what you’re doing?

```java
public class NarrowingConversionEvil {
  public static void main(String[] args) {
    int a = 200;
    byte b = (byte)a;
    System.out.println(b);
  }
}
```

NarrowingConversionEvil.java

```
% javac NarrowingConversionEvil.java
```

Assignment: Example 4

What if you don’t know what you’re doing?

```java
public class NarrowingConversionEvil {
  public static void main(String[] args) {
    int a = 200;
    byte b = (byte)a;
    System.out.println(b);
  }
}
```

NarrowingConversionEvil.java

```
% javac NarrowingConversionEvil.java
%
```

Assignment: Example 4

What if you don’t know what you’re doing?

```java
public class NarrowingConversionEvil {
  public static void main(String[] args) {
    int a = 200;
    byte b = (byte)a;
    System.out.println(b);
  }
}
```

NarrowingConversionEvil.java

```
% javac NarrowingConversionEvil.java
% java NarrowingConversionEvil
```

Assignment: Example 4

What if you don't know what you're doing?

```java
public class NarrowingConversionEvil {
  public static void main(String[] args) {
    int a = 200;
    byte b = (byte)a;
    System.out.println(b);
  }
}
```

NarrowingConversionEvil.java

```
% javac NarrowingConversionEvil.java
% java NarrowingConversionEvil
-56
```

Assignments & arithmetic operations

- We'll often use assignments like x=x+expr, where x is combined with another value (expr)
- Abbreviated syntax:

```
    x += expr     x -= expr
    x *= expr     x /= expr     x %= expr
```

- Statements x++, x−−, ++x, −−x increment/decrement the value of a variable by 1
- Value of the statement is the old (for x++) or new (for ++x) value of x

Examples

x+=5 is equivalent to x=x+5

x++ is equivalent to x+=1 and x=x+1

Assignments and operations: example

```java
public class IncOrder {
  public static void main(String[] args) {
    int i = 16;
    System.out.println(++i);
    System.out.println(i++);
    System.out.println(i);
  }
}
```

IncOrder.java

Assignments and operations: example

```java
public class IncOrder {
  public static void main(String[] args) {
    int i = 16;
    System.out.println(++i);
    System.out.println(i++);
    System.out.println(i);
  }
}
```

IncOrder.java

```
% java IncOrder
```

Assignments and operations: example

```java
public class IncOrder {
  public static void main(String[] args) {
    int i = 16;
    System.out.println(++i);
    System.out.println(i++);
    System.out.println(i);
  }
}
```

IncOrder.java

```
% java IncOrder
17
17
18
```

Summary

- Data types
- Variable declaration & assignment
- Operators

Next session

- Control structures
- Reference types

Literature

Java language basics.
`http://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html`.

Sierra, K. & Bates, B.
Head First Java.
O'Reilly Media, 2005.

Ullenboom, Ch.
Java ist auch eine Insel. (Ch. 2.3, 2.4)
Galileo Computing, 2012.