Recap

- Functions: Named code blocks
    - Can be called repeatedly
    - Can have arguments that change their behaviour
        - Arguments are accessible as variables within the body of the function
- Data types
    - Variables, literal values etc. have data types
    - Data type controls
        - How much memory is consumed
        - What can we do with the thing
    - Distinction in primitive and non-primitive types
        - For the moment: Primitive types

Primitive Data Types

Keyword	Full name	Values
`boolean`	Binary value	`true`, `false`
`byte`	1 Byte (= 8 bit)	-128 to 127
`short`	short integer (16 bit)	$-32\,768$ to $32\,767$
`int`	Integer (32 bit)	$-2\,147\,483\,648$ to $2\,147\,483\,647$
`long`	long integer (64 bit)	$-9\,223\,372\,036\,854\,775\,808$ to $9\,223\,372\,036\,854\,775\,807$
`char`	Character in UTF-16	`'\u0000'` to `'\uffff'` ($65536 = 2^{16}$ symbols)
`float`	Decimal numbers (32 bit)	$\pm 1.4 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$
`double`	Decimal numbers (64 bit)	$\pm 4.9 \times 10^{-324}$ to $\pm 1.8 \times 10^{308}$

Table: All primitive data types in Java

UNIVERSITÄT
ZU KÖLN

Session 4: Casting, Conditionals, Comments/Javadoc

Softwaretechnologie: Java 1

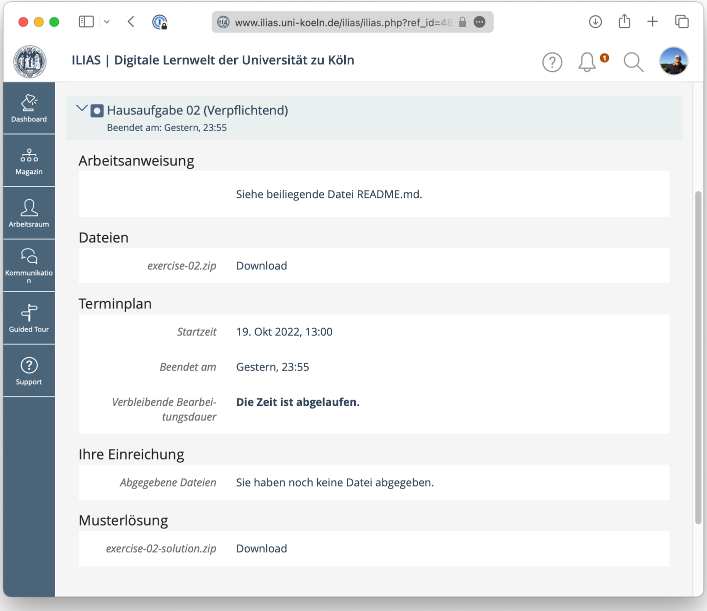
Nils Reiter
nils.reiter@uni-koeln.de

November 8, 2023

INSTITUT FÜR
DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN

Section 1

Exercise 3



www.ilias.uni-koeln.de/ilias/ilias.php?ref_id=48
ILIAS | Digitale Lernwelt der Universität zu Köln
Dashboard
Magazin
Arbeitsraum
Kommunikation
Guided Tour
Support

Hausaufgabe 02 (Verpflichtend)

Beendet am: Gestern, 23:55

Arbeitsanweisung

Siehe beiliegende Datei README.md.

Dateien

exercise-02.zip	Download

Terminplan

Startzeit	19. Okt 2022, 13:00
Beendet am	Gestern, 23:55
Verbleibende Bearbeitungsdauer	**Die Zeit ist abgelaufen.**

Ihre Einreichung

Abgegebene Dateien	Sie haben noch keine Datei abgegeben.

Musterlösung

exercise-02-solution.zip	Download

Exercise 03: isOdd(int)

```
public class Exercise03 {

^^Ipublic static void main(String[] args) {
^^I^^ISystem.out.println(isOdd(3)); // true
^^I^^ISystem.out.println(isOdd(1)); // true
^^I^^ISystem.out.println(isOdd(457483841)); // true
^^I^^ISystem.out.println(isOdd(12)); // false
^^I}

^^Istatic boolean isOdd(int number) {
^^I^^Ireturn number % 2 == 1; // shortest version, operator precedence relevant!
^^I}
^^I
}
```

Operator precedence

Section 2

Casting

Casting

- Converting from one type into another
- Explicit casting: Target type in parentheses

```
char myChar = 'a';
int myInteger = (int) myChar;
double d = (double) myInteger;
```

- Not all types can be cast into all other types
  - E.g., no casting from int to boolean
- Cast operator is an operator, i.e.: Can be used in expressions
  - `boolean b = (double) ( (int)'a' + 5 ) / 17 >= 5.0`

Implicit Casting

- If needed *and* if possible without information loss
- `double` can represent more numbers than `float`
    - `float` to `double`: No information loss
    - `double` to `float`: Potential loss
        - Explicit casting possible, use at your own risk
- `long` can represent more numbers than `short`
    - `short` to `long`: No information loss
    - `long` to `short`: Potential loss
        - Explicit casting possible, use at your own risk

Section 3

Conditionals

Conditionals

- So far: All statements are executed in sequence
- Conditionals allow specifying a condition: If it is fulfilled, a statement is executed

Conditionals

- So far: All statements are executed in sequence
- Conditionals allow specifying a condition: If it is fulfilled, a statement is executed
- Multiple forms:

```
if (EXPRESSION) STATEMENT
```

```
if (EXPRESSION) STATEMENT else STATEMENT
```

  - EXPRESSION must evaluate to a `boolean` value

== != > < <= >=

Conditionals

```
if (...) {
   ...
} else {
   ...
};
```

- So far: All statements are executed in sequence
- Conditionals allow specifying a condition: If it is fulfilled, a statement is executed
- Multiple forms:

  `if (EXPRESSION) STATEMENT`

  `if (EXPRESSION) STATEMENT else STATEMENT`

  - EXPRESSION must evaluate to a `boolean` value
- The `if` -statement is a statement, therefore:

  `if (EXP1) STATEMENT else if (EXP2) STATEMENT else STATEMENT` is also possible
- Remember: code blocks `{ ... }` are also statements

demo

Conditional Expression

- The if-statement is a statement
- Sometimes, it's useful to make such a distinction in the form of an expression
- All other operators are unitary or binary (i.e.: take one or two values)
- Ternary operator has three parts: `EXP1 ? EXP2 : EXP3`
  - EXP1 must evaluate to a boolean value, EXP2 and EXP3 must evaluate to the same type

Conditional Expression

- The if-statement is a statement
- Sometimes, it's useful to make such a distinction in the form of an expression
- All other operators are unitary or binary (i.e.: take one or two values)
- Ternary operator has three parts: `EXP1 ? EXP2 : EXP3`
  - EXP1 must evaluate to a boolean value, EXP2 and EXP3 must evaluate to the same type
- `short daysInYear = isLeapYear() ? 366 : 365;`

Switch-Statement

- Complex and embedded if-statements quickly become unreadable
- Alternative, if all if-statements compare against the same variable: `switch`-statement

Switch-Statement

- Complex and embedded if-statements quickly become unreadable
- Alternative, if all if-statements compare against the same variable: `switch`-statement

number

```
switch (EXPRESSION) {
case CONSTANT: STATEMENT; break;
case CONSTANT2, CONSTANT3: STATEMENT; break;
default: STATEMENT
}
```

5 17 23

demo

Switch-Statement

Example

```
static short daysInMonth(byte month) {
    switch(month) {
    case 2: return 28; // no break needed, because of return
    case 4: // fall through to case 11
    case 6:
    case 9:
    case 11: return 30;
    default: return 31;
  }
}
```

Section 4

Commenting

Comments

- Ignored by the compiler
- Information for us humans

Comments

- Ignored by the compiler
- Information for us humans

Two types

```
// This comment ends when the line ends

/* This comments ends with */

/*
We can include text that spans
multiple lines
*/
```

Comments

Example

```java
public class Example {

  public static void main(String[] args) {
    // stores how much users want to withdraw
    int amount = 1500;

    /* the next lines are supposed to calculate
       the third root of amount, I took the idea from
       http://www...
    */
    int temp = 3;
    amount = amount / temp;
    // TODO: Implement me!
  }
}
```

Commenting

- No fixed rules what to comment

Commenting

- No fixed rules what to comment
- Helpful: Your intentions, complex expressions, non-trivial functions
- Avoid commenting trivial things
- Keep comments up to date
- Don’t make ASCII art in comments

Javadoc

- Comments, so far: `/* ... */` and `// ...`
  - Implementation comments about your code

Javadoc

- Comments, so far: `/* ... */` and `// ...`
  - Implementation comments about your code
- New comment type: `/** ... */`
  - API comment for other programmers about a function/class/method
  - Not about specific lines, but the entire function

Javadoc

- Comments, so far: `/* ... */` and `// ...`
  - Implementation comments about your code
- New comment type: `/** ... */`
  - API comment for other programmers about a function/class/method
  - Not about specific lines, but the entire function
- API comments can be extracted to an HTML page
  - All Java classes/functions/methods have such a documentation
  - Javadoc: Integer.valueOf()

Javadoc

Javadoc

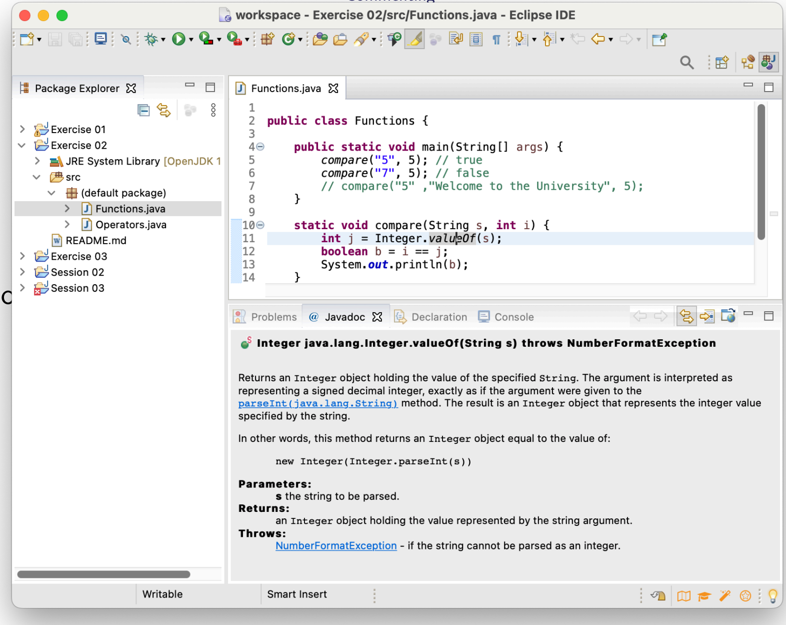
Eclipse

- Javadoc comments directly displayed by Eclipse

Javadoc

Eclipse

- Javadoc co



```
public class Functions {

    public static void main(String[] args) {
        compare("5", 5); // true
        compare("7", 5); // false
        // compare("5" ,"Welcome to the University", 5);
    }

    static void compare(String s, int i) {
        int j = Integer.valueOf(s);
        boolean b = i == j;
        System.out.println(b);
    }
```

Integer java.lang.Integer.valueOf(String s) throws NumberFormatException

Returns an `Integer` object holding the value of the specified `String`. The argument is interpreted as representing a signed decimal integer, exactly as if the argument were given to the `parseInt(java.lang.String)` method. The result is an `Integer` object that represents the integer value specified by the string.

In other words, this method returns an `Integer` object equal to the value of:

```
new Integer(Integer.parseInt(s))
```

Parameters:
s the string to be parsed.
Returns:
an `Integer` object holding the value represented by the string argument.
Throws:
NumberFormatException - if the string cannot be parsed as an integer.

Javadoc

Eclipse

- Javadoc comments directly displayed by Eclipse
- Eclipse can generate Javadoc HTML files
    - Menu > Project > Generate Javadoc ...

Section 5

Exercise