Session 4: Data types, casting, javadoc, conditionals

Softwaretechnologie: Java I

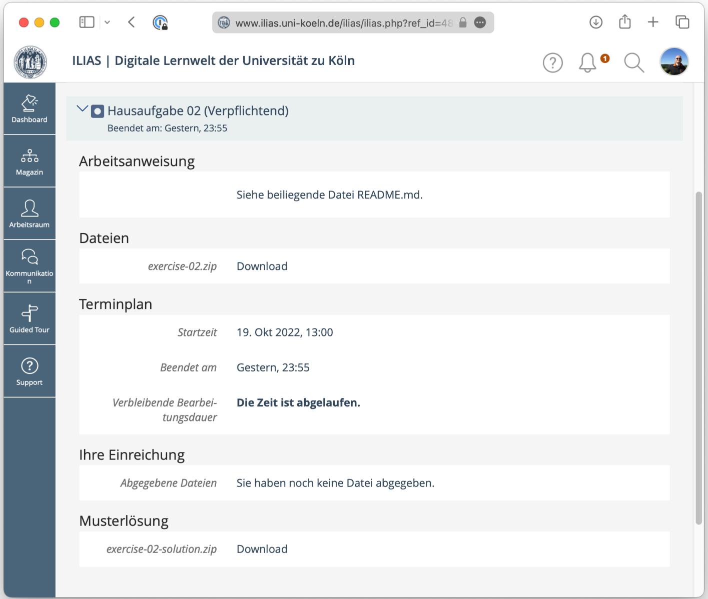
Nils Reiter
nils.reiter@uni-koeln.de

November 2, 2022

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)

```java
public class Exercise03 {

  public static void main(String[] args) {
    System.out.println(isOdd(3)); // true
    System.out.println(isOdd(1)); // true
    System.out.println(isOdd(457483841)); // true
    System.out.println(isOdd(12)); // false
  }

  static boolean isOdd(int number) {
    return number % 2 == 1; // shortest version, operator precedence relevant!
  }

}
```

Operator precedence

Section 2

Data Types, Part 2

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

Integral Data Types

Literals

- By default: full numbers within expressions are of type `int`

```
int myIntValue = 27; // literal int value assigned to an int variable
byte myByteValue = 27; // literal int value assigned to a byte variable
long myLongValue = 27; // literal int assigned to a long variable

long myLargeLongValue = 27000000000000000L;
                          // append L to enforce a long literal
long mySmallLongValue = 27L; // also works for small numbers
```

- Why can we assign an int literal to a byte/long/short variable?
  ➔Implicit casting (see below)!

Character Data

Keyword	Full name	Values
`char`	Character in UTF-16	`'\u0000'` to `'\uffff'` ($65536 = 2^{16}$ symbols)

- Characters are represented in computers by enumerating them
- American Standard Code for Information Interchange (ASCII)
    - 128 characters, including control symbols for telegraphy
    - No German Umlauts etc.

Wikipedia: ASCII

Character Data

Keyword	Full name	Values
`char`	Character in UTF-16	`'\u0000'` to `'\uffff'` ($\boxed{65536} = 2^{16}$ symbols)

- Characters are represented in computers by enumerating them
- American Standard Code for Information Interchange (ASCII)
  - 128 characters, including control symbols for telegraphy
  - No German Umlauts etc.
- Unicode: A single standard to represent *all* characters from all languages
  - $\boxed{149\,186}$ characters, including CJK ideographs
  - Complex enumeration scheme

Wikipedia: ASCII

unicode.org

Unicode 15.0 charts

Character Data

char data type

einfaches Hochkomma

- `char` represents a single character in two bytes (16 bit)
- Literal char values are written with single quotes: `char ch = 'a';`
- Unicode code points can also be used: `char ch = '\u1A0A'; //"BUGINESE LETTER NA"`
  - $1A0A_{b=16} = \underline{6666}_{b=10}$
- Integer values also possible: `char ch = 121;` (implicit cast)
- `char` is not the same as `String`

Character Data

char data type

- `char` represents a single character in two bytes (16 bit)
- Literal char values are written with single quotes: `char ch = 'a';`
- Unicode code points can also be used: `char ch = '\u1A0A'; //"BUGINESE LETTER NA"`
  - $1A0A_{b=16} = 6666_{b=10}$
- Integer values also possible: `char ch = 121;` (implicit cast)
- `char` is not the same as `String`

⚠ Not all Unicode characters can be represented as a single `char` value

- Because Unicode now defines more than 2^{16} characters
- Be aware that this might be a problem

Decimal Numbers

- Real numbers challenging for computers
- Floating-point arithmetic developed in Mesopotamia (ca. 700 BCE!)
- First used in computer by Zuse in 1937/1941

Decimal Numbers

- Real numbers challenging for computers
- Floating-point arithmetic developed in Mesopotamia (ca. 700 BCE!)
- First used in computer by Zuse in 1937/1941
- Naive idea: Two integer values, before and after decimal point
  - Wasteful and complex to implement mathematical operations

Decimal Numbers

- Real numbers challenging for computers
- Floating-point arithmetic developed in Mesopotamia (ca. 700 BCE!)
- First used in computer by Zuse in 1937/1941
- Naive idea: Two integer values, before and after decimal point
  - Wasteful and complex to implement mathematical operations
- Better idea: Represent number in scientific notation, store digits and exponent separately
  - E.g.: $\underline{123.345} \boxed{= \underline{123345}} * 10^{\boxed{-3}}$ (there are many details left out here)

Decimal Numbers in Java

Keyword	Full name	Values
`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: Floating point types

Decimal Numbers in Java

Keyword	Full name	Values
`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: Floating point types

- By default: Decimal numbers interpreted as double

Decimal Numbers in Java

Keyword	Full name	Values
`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: Floating point types

- By default: Decimal numbers interpreted as double

```java
float myFloatVariable = 3.0; // literal double, no implicit cast: compile error!
double myDoubleVariable = 3.0; // literal double
float myExplicitFloatVariable = 5.0f; // literal float value
double myDoubleVariable = 5.0f; // literal float casted into a double
```

Division, again

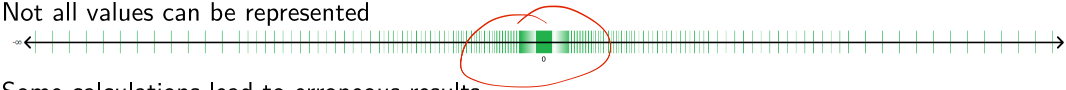
- Dividing two `int` numbers yields unexpected results (last week)
- If one number is a floating-point-number, we get decimal division

```java
int a = 7;
int bInt = 14;
System.out.println(a / bInt); // prints 0

double bFloat = 14.0;
System.out.println(7 / bFloat); // prints 0.5
```

Floating Point Complexities

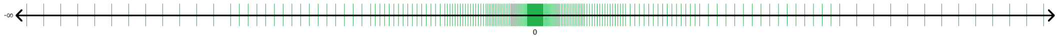
- Floating point numbers are *approximations*
  - Not all values can be represented



  - Some calculations lead to erroneous results
    E.g., `1.25f - 1.05f //yields -0.20000005`

Floating Point Complexities

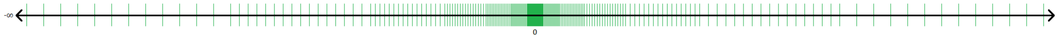
- Floating point numbers are *approximations*
  - Not all values can be represented



  - Some calculations lead to erroneous results
    E.g., `1.25f - 1.05f //yields -0.20000005`
- Java floating values have a negative zero
  E.g., `-0f` and `0f` are differently represented in memory, but defined as equal

Floating Point Complexities

- Floating point numbers are *approximations*
  - Not all values can be represented



  - Some calculations lead to erroneous results
    E.g., `1.25f - 1.05f //yields -0.20000005`
- Java floating values have a negative zero
  E.g., `-0f` and `0f` are differently represented in memory, but defined as equal
- In general: Do not use `==` with floating point numbers
  - Check if some result is 'close enough' at the expected result

Section 3

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 4

Javadoc

Javadoc

Bis Zeilenend

- 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
  - Javadoc: Integer.valueOf()

Javadoc

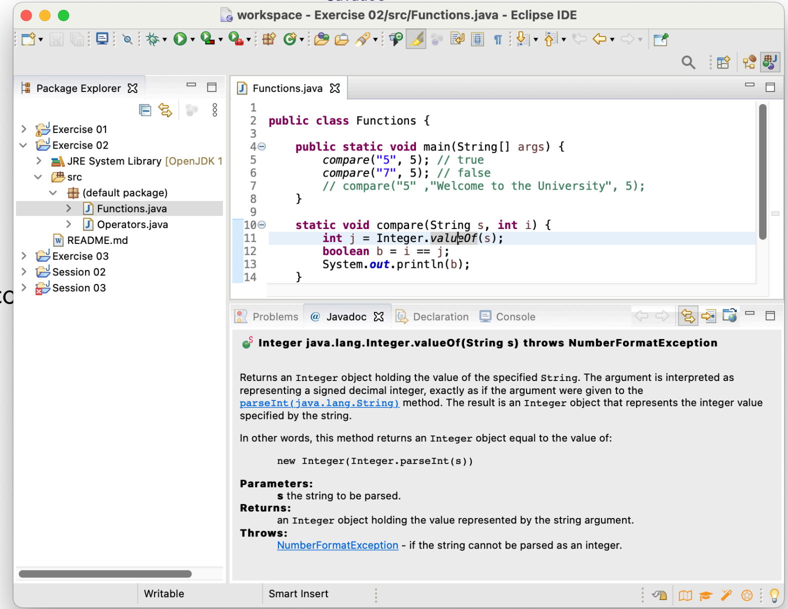
Eclipse

- Javadoc comments directly displayed by Eclipse

Javadoc

Eclipse

- Javadoc co



```java

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

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

- 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

-
  ```java
  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

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

demo

Switch-Statement

Example

```java
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 6

Exercise