Last Week

- Expressions, literals, operators, variables, identifiers, statements
- Variables are always typed and need to be declared
  - E.g., `String myString;`
- Values can be assigned to variables
  - E.g., `myString = "Guten Tag";`
- Operators, variables, literals are used in expressions
  - E.g., `5 + i` (expressions are not statements, therefore no `;`)
  - Expressions are evaluated to a defined value (of a certain type)

Exercise 2: Operators

Exercise 2: Operators

```java
public class Operators {
  public static void main(String[] args) {
    int i;
    int j;
    boolean b;
    i = 5;

    j = 5 + 5; // 10
    System.out.println(j);

    j = 5 - 5 + 5 * 2; // 10
    System.out.println(j);

    j = i - 10 * i + 10; // -35
    System.out.println(j);

    b = i + 1 > i * 1; // true
    System.out.println(b);
  }
}
```

UNIVERSITÄT
ZU KÖLN

Session 3: Functions, commenting, data types, casting

Softwaretechnologie: Java 1

Nils Reiter
nils.reiter@uni-koeln.de

October 25, 2023

INSTITUT FÜR
DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN

Section 1

Functions

Functions and Methods

- For the time being, we will use the terms function and method interchangeably
- Purpose: Code structuring
- Functions: A named code block to be defined once and called multiple times

Functions and Methods

- For the time being, we will use the terms function and method interchangeably
- Purpose: Code structuring
- Functions: A named code block to be defined once and called multiple times
- Function call: `FUNCTION_NAME ( ARGUMENTS );`
  - E.g. `System.out.println("Welcome ...");`
- Function definition: `static RETURN_TYPE FUNCTION_NAME ( ARGUMENTS ) CODE_BLOCK`

```
1 static void myFunction(String s) {
2   // some code
3 }
```

demo

Return and Return Types

- Much like expressions, functions yield a value when executed
- The type needs to be declared beforehand

  `static int bla() { ... }`: This function returns an int value

  `static boolean bla() { ... }`: This function returns a boolean value

  `static String bla() { ... }`: This function returns a String value
- Functions without return value are specified to return `void`

  ```
  static void bla() { ... }
  ```

Return and Return Types

- Much like expressions, functions yield a value when executed
- The type needs to be declared beforehand

  `static int bla() { ... }`: This function returns an int value

  `static boolean bla() { ... }`: This function returns a boolean value

  `static String bla() { ... }`: This function returns a String value
- Functions without return value are specified to return `void`

  `static void bla() { ... }`
- Within the function body
  - `return`-statement ends function, returns value

    `return 5;`

Return and Return Types

- Much like expressions, functions yield a value when executed
- The type needs to be declared beforehand

  `static int bla() { ... }`: This function returns an int value

  `static boolean bla() { ... }`: This function returns a boolean value

  `static String bla() { ... }`: This function returns a String value
- Functions without return value are specified to return `void`

  `static void bla() { ... }`
- Within the function body
  - `return`-statement ends function, returns value

    `return 5;`
- For the time being, we will additionally declare all functions as `static`
  - Until we talked about classes and objects

Function Calls in Expressions and Statements

- Function calls can be used in expressions

```
1 int x = myFunction(17) + 2345 - myOtherFunction("Hello", true);
```

Function Calls in Expressions and Statements

- Function calls can be used in expressions

```
1 int x = myFunction(17) + 2345 - myOtherFunction("Hello", true);
```

- Expressions with a semicolon are statements

```
1 myFunction(15);
2 5 + 17 / 123;
3 System.out.println("Welcome ...");
```

Arguments in Functions

- Functions can take arguments

```
static void myFunction(int x, String s, boolean b) { ... }
```

  - Arguments are declared within the function (= in the scope of the function)

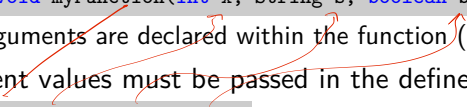
Arguments in Functions

- Functions can take arguments

```
static void myFunction(int x, String s, boolean b) { ... }
```

  - Arguments are declared within the function (= in the scope of the function)
- Argument values must be passed in the defined order when calling the function

```
myFunction(7, "Hello", true);
```



Arguments in Functions

- Functions can take arguments

```
static void myFunction(int x, String s, boolean b) { ... }
```

  - Arguments are declared within the function (= in the scope of the function)
- Argument values must be passed in the defined order when calling the function

```
myFunction(7, "Hello", true);
```

- Argument values can also be specified as expressions

```
myFunction(7 + 45, s, i < 5);
```

Section 2

Data Types

Data Types

- Java: Strong typing
- All variables and literals in Java have types
- Types are known at compile-time
- Benefit: Compiler can prevent type-related errors
  - E.g., it's a compile error to subtract a String

Primitive Data Types and Objects

Two kinds of types

- Primitive data types: Built into the language
  - Type names are reserved keywords in Java
    - I.e., it's not allowed to use them as an identifier
  - Convention: Lower cased

int
boolean

Primitive Data Types and Objects

Two kinds of types

- Primitive data types: Built into the language
  - Type names are reserved keywords in Java
    - I.e., it's not allowed to use them as an identifier
  - Convention: Lower cased
- Non-primitive data types ("reference types"): Established in the library
  - Type names are defined by library authors
  - Convention: Upper cased
  - Reference types can also be defined by us (in the form of classes, to be discussed later)

String

später

Primitive Data Types

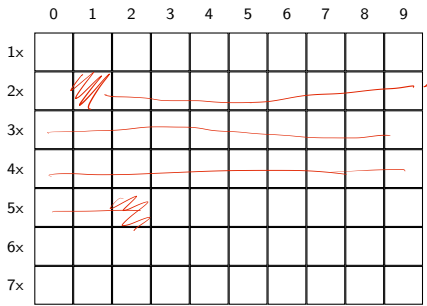
Keyword	Full name	Values
`boolean`	Binary value	`true`, `false`
`byte`	1 Byte (= 8 bit)	-128 to 127
`short`	short integer (16 bit)	$-32\,768$ to $32\,767$
`int`	Integer (32 bit)	$-2\,147\,483\,648$ to $2\,147\,483\,647$
`long`	long integer (64 bit)	$-9\,223\,372\,036\,854\,775\,808$ to $9\,223\,372\,036\,854\,775\,807$
`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

Bits and Bytes

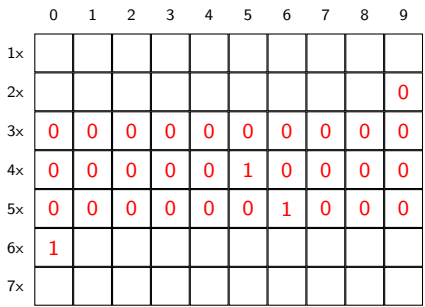
- 1 Bit: $\boxed{0}$ or $\boxed{1}$
- 1 Byte = 8 Bit: $\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}$ – $\boxed{1}\boxed{1}\boxed{1}\boxed{1}\boxed{1}\boxed{1}\boxed{1}\boxed{1}$

Primitive Types in Memory



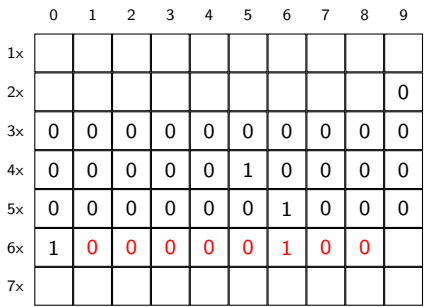

```
1 int myIntVariable = 32785;
2 // compiler knows that myIntVariable
3 // goes from bits 29 to 60
4 byte myByteVariable = 4;
5 // myByteVariable: bits 61 to 68
```

Primitive Types in Memory




```
1 int myIntVariable = 32785;
2 // compiler knows that myIntVariable
3 // goes from bits 29 to 60
4 byte myByteVariable = 4;
5 // myByteVariable: bits 61 to 68
```

Primitive Types in Memory




```
1 int myIntVariable = 32785;
2 // compiler knows that myIntVariable
3 // goes from bits 29 to 60
4 byte myByteVariable = 4;
5 // myByteVariable: bits 61 to 68
```

Data Types for Boolean Values

Keyword	Full name	Values
`boolean`	Binary value	`true`, `false`

- Occupies a single bit
- Comparison operators that produce a boolean value:
  - `5 < 7 //yields true`
  - `9 == 5 //yields false`
  - `5 <= 6 //yields true`

Data Types for Boolean Values

Keyword	Full name	Values
`boolean`	Binary value	`true`, `false`

- Occupies a single bit
- Comparison operators that produce a boolean value:
  - `5 < 7 //yields true`
  - `9 == 5 //yields false`
  - `5 <= 6 //yields true`

String s = "Hallo";

⚠ `=` and `==` are not the same thing

Data Types for Natural Numbers

Keyword	Full name	Values
`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 $9\,223\,372\,036\,854\,775\,807$

- Integral data types defined over the number of bits they occupy
- Shorter types consume less memory
- I.e.: If you know a value will never be higher than 127, use a byte
  - E.g., To store calendar weeks, human age in years(?), ...

Integral Data Types

Literals

- By default: literal numbers are of type `int` (e.g., in an expression)

```
1 int myIntValue = 27; // literal int value assigned to an int variable
2 byte myByteValue = 27; // literal int value assigned to a byte variable
3 long myLongValue = 27; // literal int assigned to a long variable
4
5 long myLargeLongValue = 27000000000000000000L;
6                         // append L to enforce a long literal
7 long mySmallLongValue = 27L; // also works for small numbers
```

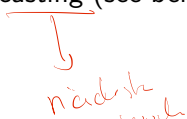
Integral Data Types

Literals

- By default: literal numbers are of type `int` (e.g., in an expression)

```
1 int myIntValue = 27; // literal int value assigned to an int variable
2 byte myByteValue = 27; // literal int value assigned to a byte variable
3 long myLongValue = 27; // literal int assigned to a long variable
4
5 long myLargeLongValue = 2700000000000000000L;
6                         // append L to enforce a long literal
7 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.
- Unicode: A single standard to represent *all* characters from all languages
  - 149 186 characters, including CJK ideographs
  - Complex enumeration scheme

Wikipedia: ASCII

unicode.org

Unicode 15.0 charts

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"`
  - $1A0A_{b=16} = 6666_{b=10}$ = ᨊ
- Integer values also possible: `char ch = 121;` (implicit cast)
- `char` is not the same as `String`

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 math

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 math
- Better idea: Represent number in scientific notation, store digits and exponent separately
  - E.g.: $123.345 = 123345 * 10^{-3}$ (there are many details left out here)

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 math
- Better idea: Represent number in scientific notation, store digits and exponent separately
  - E.g.: $123.345 = 123345 * 10^{-3}$ (there are many details left out here)
- ⚠ Floating point numbers are *approximations*, not all values can be represented

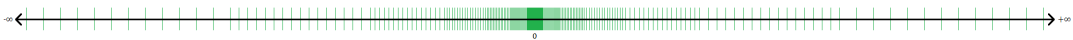


Figure: Representable numbers in floating point representation

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

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

Division, again

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

```
1 int a = 7;
2 int bInt = 14;
3 System.out.println(a / bInt); // prints 0
4
5 double bFloat = 14.0;
6 System.out.println(7 / bFloat); // prints 0.5
```

Operators

- Math operators: `+`, `-`, `*`, `/`, `%`
  - `/` behaves differently in decimal or natural mode
- Comparison operators: `<`, `<=`, `==`, `>=`, `>`
  - Yield boolean values

Operators

- Math operators: `+`, `-`, `*`, `/`, `%`
  - `/` behaves differently in decimal or natural mode
- Comparison operators: `<`, `<=`, `==`, `>=`, `>`
  - Yield boolean values
- Logical operators: `&&`, `||`, `!`
  - Yield boolean values, expect boolean input
- All operators: `https://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html`
  - Some will be introduced later in the semester

Section 3

Casting

Casting

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

```
1 char myChar = 'a';
2 int myInteger = (int) myChar;
3 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* 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

Commenting

Comments

- Ignored by the compiler
- Information for us humans

Comments

- Ignored by the compiler
- Information for us humans

Two types

```
1 // This comment ends when the line ends
2
3 /* This comments ends with */
4
5 /*
6 We can include text that spans
7 multiple lines
8 */
```

Comments

Example

```
1 public class Example {
2
3   public static void main(String[] args) {
4     // stores how much users want to withdraw
5     int amount = 1500;
6
7     /* the next lines are supposed to calculate
8        the third root of amount, I took the idea from
9        http://www...
10    */
11    int temp = 3;
12    amount = amount / temp;
13    // TODO: Implement me!
14  }
15 }
```

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

Section 5

Exercise

Section 6

Place Value Systems

Place Value Systems

"Stellenwertsysteme"

- What's the value of 712 (in the decimal system)?

Place Value Systems

"Stellenwertsysteme"

- What's the value of 712 (in the decimal system)?
  $7 * 100 + 1 * 10 + 2 * 1$

Place Value Systems

"Stellenwertsysteme"

- What's the value of 712 (in the decimal system)?
  $7 * 100 + 1 * 10 + 2 * 1$
- What's the value of x, if x has four places?

Place Value Systems

"Stellenwertsysteme"

- What's the value of 712 (in the decimal system)?
  $7 * 100 + 1 * 10 + 2 * 1$
- What's the value of x, if x has four places?
  $x_3 * 1000 + x_2 * 100 + x_1 * 10 + x_0 * 1$

Place Value Systems

"Stellenwertsysteme"

- What's the value of 712 (in the decimal system)?
  $7 * 100 + 1 * 10 + 2 * 1$
- What's the value of x, if x has four places?
  $x_3 * 1000 + x_2 * 100 + x_1 * 10 + x_0 * 1$
- Even more generic (with n being the number of digits):

$$\begin{aligned} x &= x_n * 10^n + x_{n-1} * 10^{n-1} * \cdots * x_0 * 10^0 \\ &= \sum_{i=0}^{n} x_i 10^i \end{aligned}$$

Place Value Systems

Decimal and others

- Decimal system uses base 10, but other bases could be used as well

Place Value Systems

Decimal and others

- Decimal system uses base 10, but other bases could be used as well

$$\text{value of } x \text{ with } n \text{ places, interpreted to the base } b \quad = \quad \sum_{i=0}^{n} x_i b^i$$

Examples

$$\begin{aligned} 15_{b=10} &= 1 * 10^1 + 5 * 10^0 = 15_{b=10} \\ 15_{b=8} &= 1 * 8^1 + 5 * 8^0 = 13_{b=10} \end{aligned}$$

Place Value Systems

Decimal and others

- Decimal system uses base 10, but other bases could be used as well

$$\text{value of } x \text{ with } n \text{ places, interpreted to the base } b \quad = \quad \sum_{i=0}^{n} x_i b^i$$

Examples

$$\begin{aligned} 15_{b=10} &= 1 * 10^1 + 5 * 10^0 = 15_{b=10} \\ 15_{b=8} &= 1 * 8^1 + 5 * 8^0 = 13_{b=10} \\ 15_{b=3} &= \end{aligned}$$

Place Value Systems

Decimal and others

- Decimal system uses base 10, but other bases could be used as well

$$\text{value of } x \text{ with } n \text{ places, interpreted to the base } b \quad = \quad \sum_{i=0}^{n} x_i b^i$$

Examples

$$\begin{aligned} 15_{b=10} &= 1 * 10^1 + 5 * 10^0 = 15_{b=10} \\ 15_{b=8} &= 1 * 8^1 + 5 * 8^0 = 13_{b=10} \\ 15_{b=3} &= \text{undefined symbol 5} \end{aligned}$$

Place Value Systems

Decimal and others

- Decimal system uses base 10, but other bases could be used as well

$$\text{value of } x \text{ with } n \text{ places, interpreted to the base } b \quad = \quad \sum_{i=0}^{n} x_i b^i$$

Examples

$$\begin{aligned} 15_{b=10} &= 1 * 10^1 + 5 * 10^0 = 15_{b=10} \\ 15_{b=8} &= 1 * 8^1 + 5 * 8^0 = 13_{b=10} \\ 15_{b=3} &= \text{undefined symbol 5} \\ 11_{b=2} &= 1 * 2^1 + 1 * 2^0 = 3_{b=10} \end{aligned}$$

Place Value Systems

Decimal and others

- Decimal system uses base 10, but other bases could be used as well

$$\text{value of } x \text{ with } n \text{ places, interpreted to the base } b \quad = \quad \sum_{i=0}^{n} x_i b^i$$

Examples

$$\begin{aligned}
15_{b=10} &= 1 * 10^1 + 5 * 10^0 = 15_{b=10} \\
15_{b=8} &= 1 * 8^1 + 5 * 8^0 = 13_{b=10} \\
15_{b=3} &= \text{undefined symbol 5} \\
11_{b=2} &= 1 * 2^1 + 1 * 2^0 = 3_{b=10} \\
1010_{b=2} &= 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0 = 10_{b=10}
\end{aligned}$$

Place Value Systems

Maximal Numbers

- Highest number depends on the number of places
- For 3 places
  - Decimal ($b = 10$): $999_{b=10} = 1000_{b=10} - 1$
  - Octal ($b = 8$): $777 = 511_{b=10} = 1000_{b=8} - 1$
  - Binary ($b = 2$): $111 = 7_{b=10} = 1000_{b=2} - 1$

Place Value Systems

Maximal Numbers

- Highest number depends on the number of places
- For 3 places
  - Decimal ($b = 10$): $999_{b=10} = 1000_{b=10} - 1$
  - Octal ($b = 8$): $777 = 511_{b=10} = 1000_{b=8} - 1$
  - Binary ($b = 2$): $111 = 7_{b=10} = 1000_{b=2} - 1$
- More general
  - Maximal value with n places to base b: $b^n - 1$
  - I.e.: 4 bits can distinguish 16 different states

Place Value Systems

Maximal Numbers

- Highest number depends on the number of places
- For 3 places
  - Decimal ($b = 10$): $999_{b=10} = 1000_{b=10} - 1$
  - Octal ($b = 8$): $777 = 511_{b=10} = 1000_{b=8} - 1$
  - Binary ($b = 2$): $111 = 7_{b=10} = 1000_{b=2} - 1$
- More general
  - Maximal value with n places to base b: $b^n - 1$
  - I.e.: 4 bits can distinguish 16 different states

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111