Session 2: Syntax, Variables, Operators, Functions

Softwaretechnologie: Java I

Nils Reiter
`nils.reiter@uni-koeln.de`

October 19, 2022

INSTITUT FÜR
DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN

```java
public class Demo {

  public static void main(String[] args) {

    System.out.println("Welcome to the University of Cologne!");

  }

}
```

Java Syntax

- Identifiers: Names of things
  - Case-sensitive
  - Only letters, underscore and digits, but it can't start with a digit
  - We will define identifiers ourselves
- Code blocks: Curly braces `{ ... }`
- Literals: Values that we write into the code
- `System.out.println("Welcome ...")`
  - Three identifiers, joined with a period
  - Round braces
  - A literal value
  - ➔ A function call with a single argument
- Semicolon `;`: Ends a statement/command

Java Syntax

- Identifiers: Names of things
    - Case-sensitive
    - Only letters, underscore and digits, but it can't start with a digit
    - We will define identifiers ourselves
- Code blocks: Curly braces `{ ... }`
- Literals: Values that we write into the code
- `System.out.println("Welcome ...")`
    - Three identifiers, joined with a period
    - Round braces
    - A literal value
    - ➔ A function call with a single argument
- Semicolon `;`: Ends a statement/command

- Types of statements
    - Function call

Formatting

- Java does not care about indentation or line breaks
- This:

```java
public class Demo { public static void main(String[] args) { System.out.println("Welc
```

  is a perfectly fine Java program

Formatting

- Java does not care about indentation or line breaks
- This:

```java
public class Demo { public static void main(String[] args) { System.out.println("Welc
```

  is a perfectly fine Java program
- Human programmers care about indentation and line breaks
- Programming: Dealing with complexity
  - Sensible formatting is one aspect

Formatting

- Java does not care about indentation or line breaks
- This:

  ```
  public class Demo { public static void main(String[] args) { System.out.println("Welc
  ```

  is a perfectly fine Java program
- Human programmers care about indentation and line breaks
- Programming: Dealing with complexity
  - Sensible formatting is one aspect

➔ Format your code such that it reflects the logic of the code

Variables

- Placeholders for values
- Identifier as name, but unique
- Can change over time
- Are typed: They can only hold values of one type
- Variables need to be declared before they can be used

```java
String s; // Declaration of a variable
s = "Welcome ..."; // Assignment of a
                   // value to the variable
System.out.println(s);
```

Variables

- Placeholders for values
- Identifier as name, but unique
- Can change over time
- Are typed: They can only hold values of one type
- Variables need to be declared before they can be used

```
String s; // Declaration of a variable
s = "Welcome ..."; // Assignment of a
                   // value to the variable
System.out.println(s);
```

- Types of statement(s)
  - Function call
  - Declaration
  - Assignment

Variables

- Placeholders for values
- Identifier as name, but unique
- Can change over time
- Are typed: They can only hold values of one type
- Variables need to be declared before they can be used

```
String s; // Declaration of a variable
s = "Welcome ..."; // Assignment of a
                   // value to the variable
System.out.println(s);
```

- Types of statement(s)
  - Function call
  - Declaration
  - Assignment

```
String s = "Welcome ..."; // Declaration
                          // + Assignment
System.out.println(s);
```

Variables

- Placeholders for values
- Identifier as name, but unique
- Can change over time
- Are typed: They can only hold values of one type
- Variables need to be declared before they can be used

```
String s; // Declaration of a variable
s = "Welcome ..."; // Assignment of a
                   // value to the variable
System.out.println(s);
```

```
String s = "Welcome ..."; // Declaration
                          // + Assignment
System.out.println(s);
```

- Types of statement(s)
  - Function call
  - Declaration
  - Assignment
  - Declaration+Assignment

Assignment Statements

- Assign some value to some variable
- Value can be specified literally
  - E.g. `String s = "Welcome ...";`

Assignment Statements

- Assign some value to some variable
- Value can be specified literally
  - E.g. `String s = "Welcome ...";`
- Value can be computed
  - E.g. `int i = 5 + 5;`
  - Variable `i` contains the (int) value 10
  - (Int)eger: Natural numbers from -2147483648 to 2147483647

Assignment Statements

- Assign some value to some variable
- Value can be specified literally
  - E.g. `String s = "Welcome ...";`
- Value can be computed
  - E.g. `int i = 5 + 5;`
  - Variable `i` contains the (int) value 10
  - (Int)eger: Natural numbers from -2147483648 to 2147483647
- Right side of an assignment is an *expression*

Assignment Statements

- Assign some value to some variable
- Value can be specified literally
  - E.g. `String s = "Welcome ...";`
- Value can be computed
  - E.g. `int i = 5 + 5;`
  - Variable `i` contains the (int) value 10
  - (Int)eger: Natural numbers from -2147483648 to 2147483647
- Right side of an assignment is an *expression*

- Expressions can be
  - Literal values
  - Literal values with operators

Assignment Statements

- Assign some value to some variable
- Value can be specified literally
    - E.g. `String s = "Welcome ...";`
- Value can be computed
    - E.g. `int i = 5 + 5;`
    - Variable `i` contains the (int) value 10
    - (Int)eger: Natural numbers from -2147483648 to 2147483647
- Right side of an assignment is an *expression*
    - $\text{EXPRESSION} := \text{EXPRESSION}\ \underbrace{\text{OPERATOR EXPRESSION}}_{\text{optional}}$

- Expressions can be
    - Literal values
    - Literal values with operators

More about Expressions

Expressions

- can be variables `int j = 5 + i;`

- Expressions can be
  - Literal values
  - Literal values with operators
  - Variables

More about Expressions

Expressions

- can be variables `int j = 5 + i;`
- can be nested `int j = 5 * 5 + i;`
    - Mathematical operator precedence (“Punkt vor Strich”)

- Expressions can be
    - Literal values
    - Literal values with operators
    - Variables

More about Expressions

Expressions

- can be variables `int j = 5 + i;`
- can be nested `int j = 5 * 5 + i;`
  - Mathematical operator precedence (“Punkt vor Strich”)
- can be grouped to influence preference `int j = 5 * (5 + i);`
  - Our own operator precedence according to parentheses

- Expressions can be
  - Literal values
  - Literal values with operators
  - Variables

More about Expressions

Expressions

- can be variables `int j = 5 + i;`
- can be nested `int j = 5 * 5 + i;`
  - Mathematical operator precedence (“Punkt vor Strich”)
- can be grouped to influence preference `int j = 5 * (5 + i);`
  - Our own operator precedence according to parentheses

- Expressions can be
  - Literal values
  - Literal values with operators
  - Variables

Expressions can be executed and yield a (single, clearly defined) value demo

More int-Operators

`+`	Addition	`5 + 5 //10`
`-`	Subtraction	`5 - 5 //0`
`*`	Multiplication	`5 * 5 //25`
`/`	Integer Division	`5 / 5 //1`
		`5 / 4 //1`
		`4 / 5 //0`
`%`	Modulo	`5 % 5 //0`
		`5 % 4 //1`
		`4 % 5 //4`

More int-Operators

`+`	Addition	`5 + 5 //10`
`-`	Subtraction	`5 - 5 //0`
`*`	Multiplication	`5 * 5 //25`
`/`	Integer Division	`5 / 5 //1`
		`5 / 4 //1`
		`4 / 5 //0`
`%`	Modulo	`5 % 5 //0`
		`5 % 4 //1`
		`4 % 5 //4`

All these operators operate on two `int`-values and yield an `int`-value

Comparison Operators

Symbol	Description	Example
`<`	less than	`3 < 5 //true`
`>`	greater than	`3 > 5 //false`
`==`	equal	`3 == 5 //false`

Comparison Operators

Symbol	Description	Example
`<`	less than	`3 < 5 //true`
`>`	greater than	`3 > 5 //false`
`==`	equal	`3 == 5 //false`

- Important difference
  - `==`: Comparison operator
  - `=`: Assignment operator

Comparison Operators

Symbol	Description	Example
`<`	less than	`3 < 5 //true`
`>`	greater than	`3 > 5 //false`
`==`	equal	`3 == 5 //false`

- Important difference
  - `==`: Comparison operator
  - `=`: Assignment operator
- New type: `boolean`
  - Only two possible values: `true` or `false`

Comparison Operators

Symbol	Description	Example
`<`	less than	`3 < 5 //true`
`>`	greater than	`3 > 5 //false`
`==`	equal	`3 == 5 //false`

- Important difference
  - `==`: Comparison operator
  - `=`: Assignment operator
- New type: `boolean`
  - Only two possible values: `true` or `false`

More operators

Variables and Scope

- Most variables have limited validity: Their scope
- Code blocks define scope boundaries
- Scope is nested: We can access upwards, but not downwards

Variables and Scope

- Most variables have limited validity: Their scope
- Code blocks define scope boundaries
- Scope is nested: We can access upwards, but not downwards

```java
public class Scope {

  public static void main(String[] args) {
    int a = 5;
    int b = 17;
  }
}
```

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: `RETURN_TYPE FUNCTION_NAME ( ARGUMENTS ) CODE_BLOCK`

```java
void myFunction(String s) {
  // some code
}
```

demo

Return and Return Types

- Much like expressions, functions yield a value when executed
- The type needs to be known 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 known 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;`

Function Calls in Expressions and Statements

- Function calls can be used in expressions

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

Function Calls in Expressions and Statements

- Function calls can be used in expressions

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

- Expressions with a semicolon are statements

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

- Types of statement(s)
  - Expression + `;`
    - Function call
    - Assignment
    - ...
  - Declaration
  - Decl. + Expression + `;`

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 1

Exercise

Exercise 02

- Fill in operators such that the expected result is computed
- Write functions
  - to calculate x^3
  - to compare a String and an int value