UNIVERSITÄT
ZU KÖLN

Session 2: Syntax, Variables, Operators, Functions

Softwaretechnologie: Java 1

Nils Reiter
nils.reiter@uni-koeln.de

October 18, 2023

INSTITUT FÜR
DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN

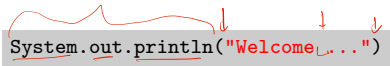
```java
public class Demo {
  public static void main(String[] args) {
    System.out.println("Welcome to the University of Cologne!");
  }
}
```

Statements

- Sequences of statements: A program
- Each statement ends with a semicolon `;`
- Different kinds of statements
  - Function call: `System.out.println("Welcome ...");`
    - A pre-defined subroutine/mini program that does something
- Statements can be grouped into code blocks with curly braces `{ ... }`

Statements

Function Calls



```
System.out.println("Welcome␣...")
```

- Three identifier, joined with a period
- Round braces
- A literal value
- A function call with a single argument

Java Syntax

- Identifiers: Names of
  - Case-sensitive
  - Only letters, underscore and digits
    - Cannot start with a digit
  - We will define identifiers ourselves

Java Syntax

- Identifiers: Names of
  - Case-sensitive
  - Only letters, underscore and digits
    - Cannot start with a digit
  - We will define identifiers ourselves
  - Which of the following is a legal identifier?
    - `hello`
    - `hällö`
    - `this_is_an_identifier_or_is_it?`
    - `king-charles-5`
    - `3doorsdown`

Java Syntax

- Identifiers: Names of
  - Case-sensitive
  - Only letters, underscore and digits
    - Cannot start with a digit
  - We will define identifiers ourselves
  - Which of the following is a legal identifier?
    - `hello`
    - `hällö`
    - `this_is_an_identifier_or_is_it?`
    - `king-charles-5`
    - `3doorsdown`
- Literals: Values that we write into the code
  - E.g., `"Welcome ..."`

Formatting

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

```
1 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:

```
1 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:

```
1 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 must be unique
- Can change over time
- Are typed: They can only hold values of one type
- Need to be declared before they can be used
- Can be used instead of literal values

Variables

- Placeholders for values
- Identifier as name, but must be unique
- Can change over time
- Are typed: They can only hold values of one type
- Need to be declared before they can be used
- Can be used instead of literal values
- New kinds of statements
  - Declaration of a variable: `String s;`
  - Assignment of a value to a variable: `s = "Welcome ...";`

Variables

- Placeholders for values
- Identifier as name, but must be unique
- Can change over time
- Are typed: They can only hold values of one type
- Need to be declared before they can be used
- Can be used instead of literal values
- New kinds of statements
  - Declaration of a variable: `String s;`
  - Assignment of a value to a variable: `s = "Welcome ...";`

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

Variables

- Placeholders for values
- Identifier as name, but must be unique
- Can change over time
- Are typed: They can only hold values of one type
- Need to be declared before they can be used
- Can be used instead of literal values
- New kinds of statements
  - Declaration of a variable: `String s;`
  - Assignment of a value to a variable: `s = "Welcome ...";`
  - Declaration and assignment in one statement: `String s = "Welcome ..."`

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

Variables

- Placeholders for values
- Identifier as name, but must be unique
- Can change over time
- Are typed: They can only hold values of one type
- Need to be declared before they can be used
- Can be used instead of literal values
- New kinds of statements
  - Declaration of a variable: `String s;`
  - Assignment of a value to a variable: `s = "Welcome ...";`
  - Declaration and assignment in one statement: `String s = "Welcome ..."`

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

```
1 String s = " ..."; // Declaration
2                    // + Assignment
3 System.out.println(s); // Func. call
```

Assignment Statements

Assignment Statements

```
String myText = "Welcome ... " ;
```

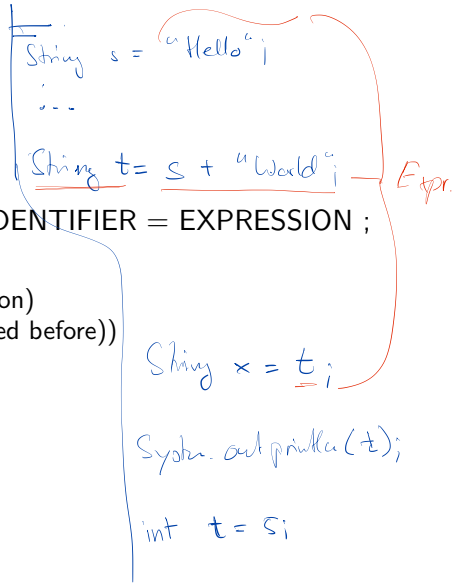
- Assign some value to some variable
- Right-hand side (RHS): The value
  - E.g., a literal value: `String s = "Welcome ...";`

Assignment Statements

```
String myText = "Welcome ... " ;
```

- Assign some value to some variable
- Right-hand side (RHS): The value
  - E.g., a literal value: `String s = "Welcome ...";`
  - In general, RHS is an expression

Expressions



- Structure of an decl+assignment statement: TYPE IDENTIFIER = EXPRESSION ;
- Different kinds of expressions
  - Literal values are expressions (`"Welcome"` is an expression)
  - Variables are expressions (`s` is an expression (if declared before))

Expressions

- Structure of an decl+assignment statement: TYPE IDENTIFIER = EXPRESSION ;
- Different kinds of expressions
  - Literal values are expressions (`"Welcome"` is an expression)
  - Variables are expressions (`s` is an expression (if declared before))
  - Expressions combined with operators are expressions
    - Operators: comparison, mathematical computation, and many more
    - E.g., `"Hi"` + `"Everyone"` is an expression
      (because the plus-operator is defined for strings)

More Expressions

All the following are expressions:

- `5 + 7`
- `5 + i` (if a variable `i` is defined and of the correct type)
- `5 + 5 * i` (if a variable `i` is defined and of the correct type)
  - Mathematical operator precedence ("Punkt vor Strich")
- `(5 + 5) * i` (if a variable `i` is defined and of the correct type)
  - We can use parentheses to influence the order in which things are computed

- Expressions can be executed and yield a (single, clearly defined) value
- The result of the expression is assigned (if the expression is part of an assignment)

demo

Terminology

- Please verify that your neighbour knows the following terms
  - Expression
  - Literal
  - RHS
  - Operator
  - Variable
  - Identifier
  - Statement

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

```
1  public class Scope {
2
3    public static void main(String[] args) {
4      int a = 5;
5      int b = 17;
6
7      int c = a + 45;
8      System.out.println(c);
9      {
10       int d = b - 10;
11       System.out.println(d);
12
13     }
14     int d = b - 10;
15     System.out.println(d);
16
17   }
18 }
```

String d = "Hallo";

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`

```
1 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 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

```
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 1

Exercise