Last Week

- Expressions, literals, operators, variables, identifiers, statements, data types
- 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)
- Data types: `int`, `boolean`, `char`, ...
  - Operators can be used to manipulate values of certain types

Exercise 2: Operators

Exercise 2: Operators

```
1  public class Operators {
2    public static void main(String[] args) {
3      int i;
4      int j;
5      boolean b;
6      i = 5;
7
8      j = 5 + 5; // 10
9      System.out.println(j);
10
11     j = 5 - 5 + 5 * 2; // 10
12     System.out.println(j);
13
14     j = i - 10 * i + 10; // -35
15     System.out.println(j);
16
17     b = i + (1 > i) * 1; // true
18     System.out.println(b);
19   }
20 }
```

false

UNIVERSITÄT
ZU KÖLN

Session 3: Variables 2, Functions, Comments

Softwaretechnologie: Java I

Nils Reiter
nils.reiter@uni-koeln.de

October 30, 2024

INSTITUT FÜR
DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN

Section 1

Variables and Data Types, part two

Operators

13 / 5 = 2
13 % 5 = 3

- Math operators: `+`, `-`, `*`, `/`, `%` (modulo)
  - `/` behaves differently in float or int mode
- Comparison operators: `<`, `<=`, `==`, `>=`, `>`
  - Yield boolean values

13 >= 5 = true

Operators

- Math operators: `+`, `-`, `*`, `/`, `%`
  - `/` behaves differently in float or int 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

Math 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`

Math 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`

- Yield a boolean value
- Important difference
  - `==`: Comparison operator
  - `=`: Assignment operator

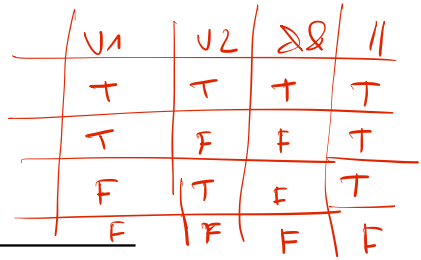
Comparison Operators

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

- Yield a boolean value
- Important difference
  - `==`: Comparison operator
  - `=`: Assignment operator

More operators

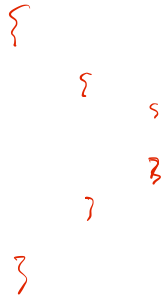
Logical Operators



Symbol	Description	Example
`&&`	logical and	`true and false //false`
`||`	logical or	`true or false //true`
`!`	negation	`!true //false`

Variables and Scope

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

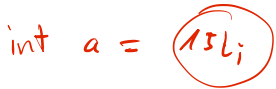


Variables and Scope

- Most variables have limited lifetime: 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 }
```

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
- Casting is an operator ➔ usable in expressions
  - `boolean b = (double) ( (int)'a'+ 5 ) / 17 >= 5.0`

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
- Casting is an operator ➔ usable in expressions
  - `boolean b = (double) ( (int)'a'+ 5 ) / 17 >= 5.0`

Things usable in expressions:

- Literal values `5`
- Variables `a`
- Operators `a + 5`
- Parentheses `( )`
- Casting `(double) 5`

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 2

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 talk about classes and objects

Function Calls in Expressions and Statements

- Function calls can be used in expressions

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

int int

Function Calls in Expressions and Statements

- Function calls can be used in expressions

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

- Expressions with a semicolon are statements

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

Function Calls in Expressions and Statements

- Function calls can be used in expressions

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

- Expressions with a semicolon are statements

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

Things usable in expressions:

- Literal values `5`
- Variables `a`
- Operators `a + 5`
- Parentheses `( )`
- Casting `(double) 5`
- Function calls `greet("Nils", "de")`

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);
```

The Main Function

```
1 public static void main(String[] args) {
2     // ...
3 }
```

- All programs contain at least one main function
- Entry point to your program
  - When you start your program this function is “called”
- A regular function, just with a special name

Section 3

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

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

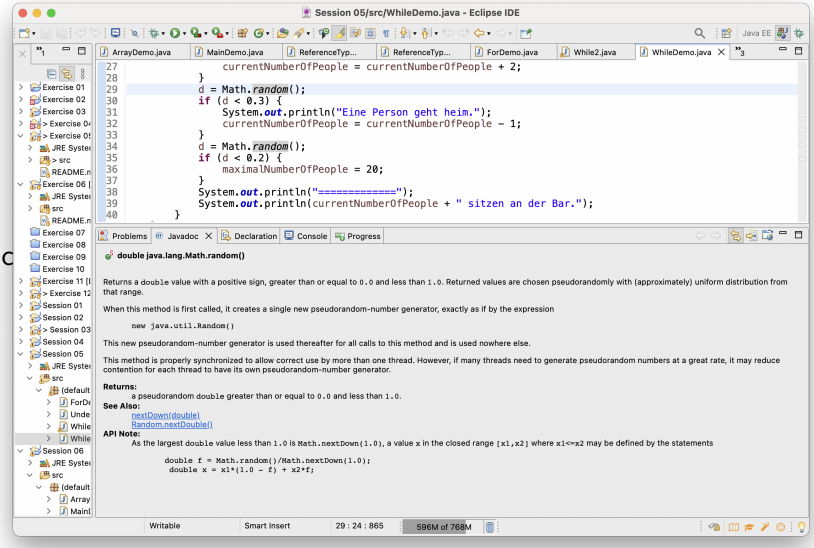
Eclipse

- Javadoc comments directly displayed by Eclipse

Javadoc

Eclipse

- Javadoc c



Javadoc

Eclipse

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

Section 4

Exercise

Section 5

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