If-statements and the Java standard library

Lecture 2 of TDA 540
Object-Oriented Programming

Jesper Cockx

Fall 2018

Chalmers University of Technology — Gothenburg University

Lab sessions

First lab sessions: today at 15:15 and tomorrow at 8:00 and 10:00.

First deadline: 20/9 (next week).

Start on time

It always takes longer than you expect, even when you take into account Hofstadter's Law.[1]

[1] Douglas Hofstadter (1979): *Gödel, Escher, Bach: an Eternal Golden Braid.*

Last time on TDA540...

Last lecture

- Programming languages, machine code, and compilers
- Algorithms in pseudocode
- Basic types: `int`, `double`, `String`, ...
- Using IntelliJ to create a new project
- Compile-time errors and run-time errors

Naming conventions in Java

Classes Upper case

```
String
public class MyClass { ... }
```

Variables and methods Lower / Camel case

```
String name;
int iAmACamelCase;
```

Constants All caps + `final` keyword

```
final int DAYS_IN_YEAR = 365;
final int ONE_DOZEN = 12;
```

`Math.PI`, `Integer.MAX_VALUE`

Type systems

Type systems

Why do we have type systems?

- Help organizing and structuring your programs
- Communicate the expected inputs and outputs of a method
- Give an error message when using an operation that is not permitted

Type systems

Static type systems raise error messages at compile-time.

Dynamic type systems raise error messages at run-time.

Java uses a static type system, allowing you to spot mistakes early on.

Type conversion: implicit

```
double value1 = 1;        // x = 1.0
double value2 = 3 + 0.5   // value2 = 3.5
```

Order of implicit conversion: `byte` → `short` → `int` → `long` → `float` → `double`

Type conversion: type cast

A type cast converts a value from one type to another or a library method.

Syntax: `(type) value`

```
byte value3 = (byte) 400;
    // value4 = -112
int value4 = (int) (0.8 + 0.9);
    // value3 = 1
```

Warning: casting to `int` always rounds down.

Type conversion: library method

Many library methods allow you to convert from one type to another:

```
int value5 = (int) Math.round(0.8 + 0.9)
      // value5 = 2
String value6 = Integer.toString(123);
      // value6 = "123"
int value7 = Integer.valueOf("123");
      // value7 = 123
```

The Java standard library

The Java standard library

https://docs.oracle.com/javase/10/docs/api

How to read a method declaration

```
static double pow (double num, double power)
```

↑ return type

↑ name

↑ parameters (type + name)

Example usage:

```
double x = Math.pow(12.5, 3.0);
```

Static vs. non-static methods

`static` methods belong to a class such as `Math`:

```
double myNumber = 500.00;
double mySquareRoot = Math.sqrt(myNumber)
```

Non-`static` methods belong to a specific object such as a string:

```
String myString = "Some string";
int length = myString.length();
```

Some important library classes

Module `java.base`, class `java.lang.Math`:

`static final double PI`	mathematical constant π
`static final double E`	mathematical constant e
`static double abs(double a)`	absolute value
`static double exp(double a)`	exponential function e^a
`static double log(double a)`	natural logarithm $ln(a)$
`static double min(double a, double b)`	minimum of two numbers
`static double max(double a, double b)`	maximum of two numbers
`static double pow(double a, double b)`	exponential a^b
`static long round(double a)`	round to closest integer
`static double sqrt(double a)`	square root $\sqrt{a}$
`static double sin(double a)`	sine function $\sin(a)$
`static double cos(double a)`	cosine function $\cos(a)$
`static double tan(double a)`	tangent function $\tan(a)$
`static double toDegrees(double angrad)`	convert from radians to degrees
`static double toRadians(double angdeg)`	convert from degrees to radians
`static double random()`	generate a random number in $[0.0, 1.0)$

Some important library classes

Module java.base, class java.lang.String:

`int length()`	length of a string
`char charAt(int index)`	character at the specified index
`String replace(char oldChar, char newChar)`	create new string with `oldChar` replaced by `newChar`
`String toUpperCase()`	create new string with all characters converted to UPPER CASE
`String toLowerCase()`	create new string with all characters converted to lower case
`String substring(int beginIndex, int endIndex)`	take substring from `beginIndex` to `endIndex-1`

All these methods are non-static:

```
String myString = "Some string";
myString.substring(2,8); // "me str"
```

Reading command-line input with `Scanner`

```java
import java.util.Scanner;

public class HelloWorld {

    public static void main(String[] args) {
        Scanner userInput = new Scanner(System.in);
        System.out.print("Please enter your name: ");
        String name = userInput.next();
        System.out.println("Hello, " + name + "!");
    }

}
```

String formatting with `String.format`

```
static String format(String format,
    Object... args)
```

Any parts starting with a `%` sign are replaced by the corresponding argument:

- `"%s"` : a string
- `"%d"` : an integer number
- `"%f"` : a real number in decimal notation
- `"%e"` : a real number in scientific notation

Example of string formatting

```
String msg = String.format(
  "%s | %d | %f | %e",
  "one", 2, 3.0, 456.789);
System.out.println(msg);
```

prints out...

```
one | 2 | 3.000000 | 4.567890e+02
```

More about string formatting

You can give the desired precision of a floating-point number by `"%.nf"`:

```
String.format("%.5f", Math.PI); // "3.14159"
```

You can give the desired width by `"%ns"` (or `"%nd"`, ...):

```
String.format("|%20s|", "short");
               // "|               short|"
```

You can align left instead by adding a minus sign `-`:

```
String.format("|%-5d|", 1);   // "|1    |"
```

Application: printing a table

```
final String HEADER_FORMAT = "| %-10s | %-10s |\n";
final String DATA_FORMAT   = "| %10.2f | %10d |\n";
System.out.printf(HEADER_FORMAT, "Price", "Amount");
System.out.printf(DATA_FORMAT, 12.345, 10);
System.out.printf(DATA_FORMAT, 0.111, 5);
```

prints out...

```
| Price      | Amount     |
|      12.35 |         10 |
|       0.11 |          5 |
```

Quiz: Variables and primitive types in Java

Instructions:

- Go to `kahoot.it` on your phone or laptop
- Enter the code on the big screen

15 min. break

If-statements

Reminder: two kinds of instructions

- *Atomic* instructions (e.g. increase *x* by 1, wait 1 second, launch missile, ...)
- *Control* instructions:

Sequence	First do *x*, then do *y*
Choice	If *x* is true, then do *y*, else do *z*
Iteration	As long as *x* is true, repeat *y*
Jump	Continue from point *x*
...	

Reminder: two kinds of instructions

- *Atomic* instructions (e.g. increase *x* by 1, wait 1 second, launch missile, ...)
- *Control* instructions:

  Sequence First do *x*, then do *y*

  Choice If *x* is true, then do *y*, else do *z*

  Iteration As long as *x* is true, repeat *y*

  Jump Continue from point *x*

  ...

Basic structure of an if-statement

```
if ( condition ) {
  // code here is executed when
  // the condition is *true*
} else {
  // code here is executed when
  // the condition is *false*
}
```

(`else`-part can be omitted)

Comparing numbers

You can compare numbers using `<`, `>`, `<=` ($\leq$), `>=` ($\geq$), `==` ($=$), and `!=` ($\neq$):

```java
int number = JOptionPane.showInputDialog(
  "Please enter a number");
if (number <= 9000) {
  JOptionPane.showMessageDialog(null,
    "That is not a very big number...");
} else {
  JOptionPane.showMessageDialog(null,
    "Wow, that's a big number!");
}
```

Comparing strings

Warning: never use `==` to compare two `String`s!

Instead, use the method

```
boolean equals(Object anObject):
```

```
String oneString = "123" + "456";
String anotherString = "12" + "34" + "56";
if ( oneString.equals(anotherString) ) {
   ...
else {
   ...
}
```

Sequential if-statements

```
if ( condition1 ) {
  // code here is executed when
  // condition1 is true
} else if ( condition2 ) {
  // code here is executed when
  // condition1 is false and
  // condition2 is true
} else if ( condition3 ) {
  ...
} else {
  // code here is executed when
  // ALL conditions are false
}
```

Nested if-statements

```
if ( condition1 ) {
  if ( condition2 ) {
    // executed if condition1 AND
    // condition2 are true
  else {
    // executed if condition1 is true
    // and condition2 is false
  }
} else {
  // executed if condition1 is false
}
```

Live coding: leap years

Assignment: Write a program that calculates whether a given year is a leap year.

Composing multiple conditions

You can compose multiple conditions using `&&` (and), `||` (or), `!` (not), and `^` (exclusive or):

```
if ( (year % 4 == 0 && year % 100 != 0)
     || year % 400 == 0 ) {
  System.out.println(year
      + " is a leap year");
else {
  System.out.println(year
      + " is not a leap year");
}
```

Side note: De Morgan’s laws

```
!(x && y) == !x || !y

!(x || y) == !x && !y
```

Order of operations

Operator	Description
++, --	increment, decrement
!	logical not
*, /	multiplication, division
+, -	addition, subtraction
<, >, <=, >=	number comparisons
==, !=	equality, inequality
^	exclusive or
&&	logical and
||	logical or

Order of operations: example

```
year % 4 == 0 && year % 100 != 0
    || year % 400 == 0
```

has the same meaning as

```
(((year % 4) == 0) && ((year % 100) != 0))
    || ((year % 400) == 0)
```

Tip: when in doubt, write more parenthesis!

Short-circuit evaluation of `&&` and `||`

Evaluation of `&&` and `||` in Java is lazy:

- If `x == true`, Java will *never* compute `y` in the expression `x || y`
- If `x == false`, Java will *never* compute `y` in the expression `x && y`

Application: safe division

```
int num = ...; int denom = ...;
if ((denom != 0) && (num / denom > 10.5) {
   ...
}
```

Application: input validation

You can use an `if`-statement to check if the input makes sense:

```java
int number = JOptionPane.showInputDialog(
  "Enter a positive number");
if ( number < 0 ) {
  JOptionPane.showMessageDialog(null,
    "Error: not a positive number");
} else {
  // rest of the program goes here
}
```

Application: input validation

Alternative: use `System.exit()` to shut down the program:

```
int number = JOptionPane.showInputDialog(
  "Enter a positive number");
if ( number < 0 ) {
  JOptionPane.showMessageDialog(null,
    "Error: not a positive number");
  System.exit(0);
}
// rest of the program goes here
```

What's next?

Next lecture: `while` and `for` loops

To do:

- Read the book:
  - Today: parts of chapter 2 and chapter 3
  - Next lecture: chapter 4
- Go to the lab sessions
- Find a partner for the labs
- Register your group on Fire