Introduction to Programming

René Thiemann

Institute of Computer Science
University of Innsbruck

WS 2008/2009

Outline

Outline

Organization

- alternating lectures (HS 11, me) and exercises (RR 21, Berthold Agreiter)
- one has to register twice for both the lecture and the exercises until Oct 3, 12am and until Oct 10, 8am
- exercise course
  - new exercises are available each Tuesday after the lecture
  - all exercises have to be solved before the next exercise course
  - each exercise is presented by a randomly selected student
  - these presentations determine the grade of the exercise course
  - the solution to exercises which are not discussed will be made available
- passing the exercise course is required to register for final exam
- final exam is combined exam of this lecture and the lecture Einführung in die Informatik of Barbara Weber
- details about grading of whole module: see lecture of Barbara Weber
- more details are available at the website of this lecture

Literature

- Klaus Echtle and Michael Goedicke,
  Lehrbuch der Programmierung mit Java, dpunkt.verlag, 2000
- Judy Bishop, Java Gently, Addison Wesley, 2001
- Christian Ullenboom, Java ist auch eine Insel, Galileo Computing, 2007, available online

Prerequisites

- interest in writing computer programs
- access to a computer where Java is installed

Selection of Topics

- overview of computer programs
- fundamental ingredients of imperative programming languages
- object-oriented programming
- programming with dynamic data-structures
- ...

Outline

Abilities of a computer

What it can do

- process large amounts of data in few time
- handle optimization and search problems
- ...
- internally: execute basic instruction... over and over again

What it cannot do

- being inventive

$\Rightarrow$ no guessing in case of errors

$\Rightarrow$ basic instructions must be precise
(a computer cannot guess the intended meaning)

$\Rightarrow$ program (code): precise description consisting of basic instruction to solve a problem

Good properties of code

- sound & complete: computes the correct result for any valid input
- documented: important for outside use: what are valid inputs and what is the computed result
- human readable: for maintenance and future extensions of the code
- robust: code is aware of invalid inputs and handles these (e.g., by informing the user why the input is invalid)
- efficient: result is returned quickly and without using much memory
- parallelizable: code is executed faster if computer has many processors (dual-core, etc.)

this lecture focusses only on the first three items

Format of the code

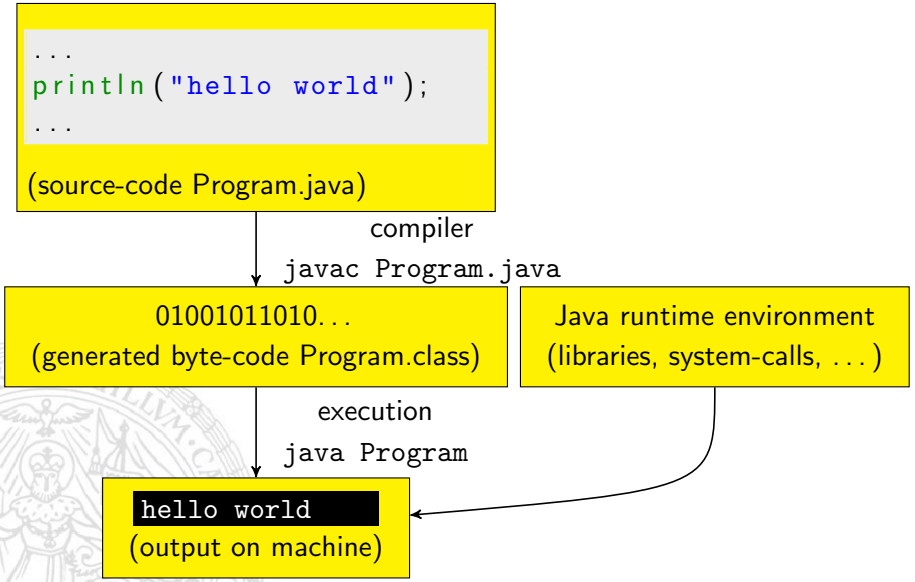
- code should be human readable, source-code: `print("hello world");`
- code should be machine-readable, byte-code: 0111010001...

⇒ conflict in format

Solution: compiler

1. human writes source-code in high-level language (Java, C++, C#, Haskell, Prolog, ...)
2. compiler translates source-code into byte-code
3. machine executes byte-code

From source-code to output



Syntax and semantic of programming languages

- Syntax: defines what a valid program is

⇒ if a program has a syntactical error then it is refused by the compiler

- Semantic: defines the meaning of a valid program

⇒ if a program has a semantical error then the compiled byte-code will produce incorrect results

- Syntax-checks can be done automatically (without knowledge of the purpose of the program)
- Semantic-checks are only possible w.r.t. the purpose of the program

Example (Purpose: output "hello world")

```
print("hello world"; // syntax error
print("hello word"); // semantic error
```