Programmieren II - Java

Introduction to Java

Alexander Fraser

fraser@cl.uni-heidelberg.de

(Contains material from: T. Bögel, K. Spreyer, S. Ponzetto, M. Hartung)

April 23, 2014

Goals of this session

- Details, getting a Schein
- Introduction to Java, why learn Java?
- How to write a “hello world” program in java

Details

Time and Place

- Wed, 14:15–15:45
  Place: INF 306 / SR 19
- Thurs, 14:15–15:45
  Place: INF 328 / SR 25

Exceptions so far

- Three Thursday holidays
- Tag der Arbeit : 01.05.2014
- Christi Himmelfahrt : 29.05.2014
- Fronleichnam : 19.06.2014
- Possible business trip: 25.06-26.06.2014

Tutorium

Tutoren

- This is still being worked out, expect an update very soon.

Ziele und Inhalte

- Wiederholung und Einübung
- Korrektur der Übungsaufgaben

Prerequisites

Programming Experience?

- Ideally: $\geq$ 2nd Semester
- Programmieren I
- Alternatively: good programming experience

Kursziele

Zusammenfassung

- Erweiterung des Wissens aus “Programmieren I” und praktischer Programmierkenntnisse
- *Objektorientierte* Sicht auf Probleme
- Prinzipien von gutem Softwaredesign
- Schöner und verständlicher Code
- Einführung in das systematische Testen von Code

Kriterien zum erfolgreichen Besuch

Formale Kriterien

- Regelmäßige Anwesenheit & Mitarbeit
- Erfolgreiches Bearbeiten der Übungsaufgaben (65% der Punkte)
- Klausur

How this should work

- *Aktive* Teilnahme und Diskussionen während der Vorlesung
- Fragen, bis *alles* verstanden ist
- Bei Problemen: Feedback, Mail, Sprechstunde
- Kommentare & Dokumentation!
- Schreibt euren *eigenen* Code!

Vorlesungssitzungen

Vorlesungssitzungen bestehen aus:

- Theorie
- Praktischer Anwendung in Form von Implementierung & Diskussionen

Overall concepts

- Building on Prog I:
- Java syntax
- OOP (objected oriented programming)
- Software design

Schedule

Time	Content
Week 01	Introduction & elementary concepts of Java
Week 02	Java language specification, syntax
Week 03,04	Introduction to OOP & remaining java syntax
Week 05	OOP & exceptions
Week 06	Input/output
Week 07	Collections & more advanced OOP
Week 08+	To be determined

Prüfungsmodalitäten

- Klausur: 24.07.2014 (to be confirmed)
- 7-8 Übungsaufgaben

Übungsaufgaben

- Bearbeitungszeit: 1 – 2 Wochen
- **Einzelarbeit**
- Gegenseitige Bewertung von Übungsaufgaben
- Keine verspätete Abgabe nach Deadline
  $\rightarrow$ für Verlängerung *vorher* melden!

Kontaktinformationen

Büro und Sprechstunde

Sprechstunde: Dienstag, 15-16 Uhr (bitte anmelden per E-Mail)
Im Neuenheimer Feld 325
Diese Woche: Raum 122 (1. Stock)

Email

- Email: `fraser@cl.uni-heidelberg.de`

Goals of this session

- ☑ Details, getting a Schein
- ☐ Introduction to Java, why learn Java?
- ☐ How to write a “hello world” program in java

Java in buzzwords [1]

Java as a programming language

- Simple
- Object oriented
- Distributed
- Multithreaded
- Dynamic
- Architecture neutral
- Portable
- High performance
- Robust
- Secure

[1] http://www.oracle.com/technetwork/java/langenv-140151.html

Vom Programm zum Binärcode

Problem

- Prozessoren sprechen nur in Bits (Maschinensprache/Maschinencode): 00110101
- Menschen sprechen nicht in Bits (und können auch nur schlecht in Bits programmieren)

$\Rightarrow$ Programme werden in höheren Programmiersprachen (Java, Python) geschrieben und dann eine niedrigere Sprache (Assembler, Maschinencode, Bytecode) übersetzt

From the Program to Machine Code I

Differences Python - Java

- Python is typically viewed as interpreted
  (Translation and execution at once)
- Java is typically viewed as (bytecode-)compiled
  (Translation first, execution later)
- But, actually, python is also bytecode-compiled (.pyc files)

Compiled vs. Interpreted

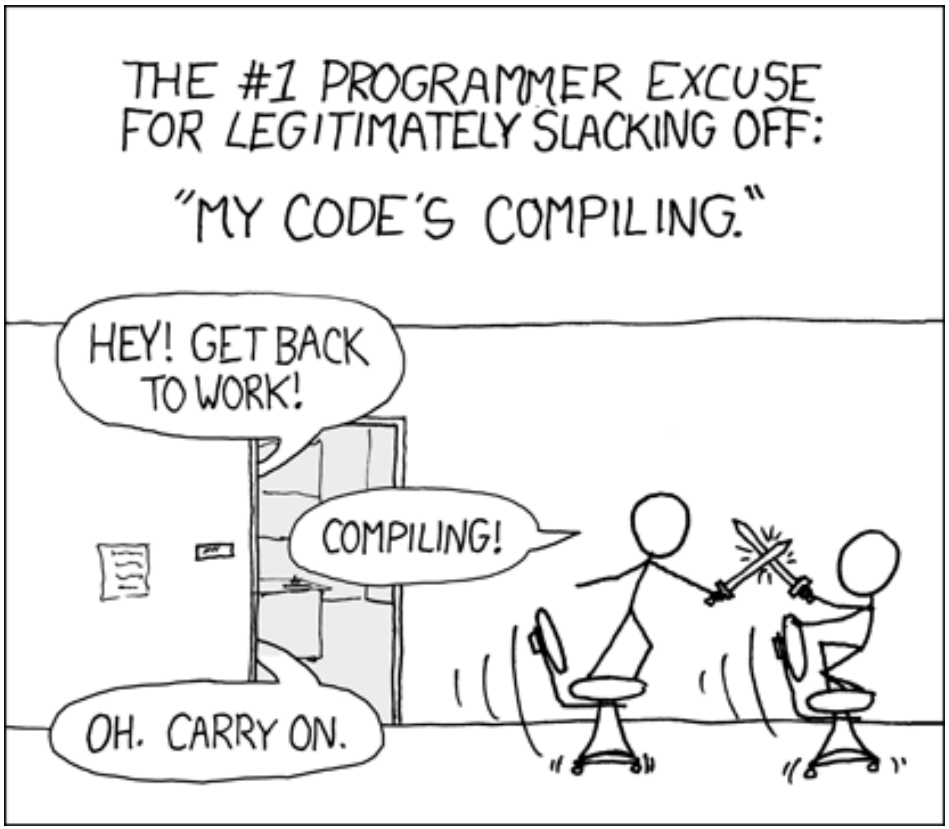
- Compiled programs run faster, but are platform-dependent
- Interpreted programs tend to be platform-independent
- Java bytecode is theoretically (and often pratically) platform-independent

From the Program to Machine Code II

Python vs Java - my view

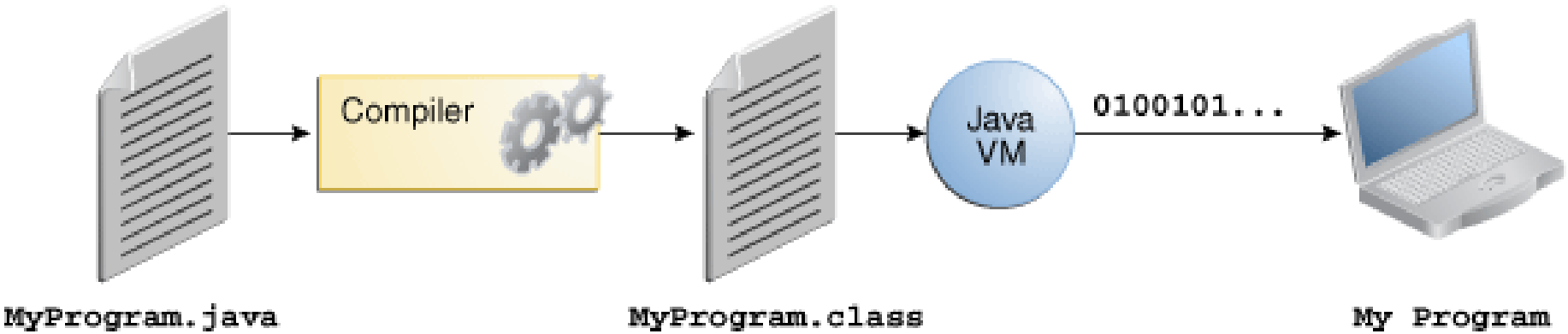
- Quick prototyping, data munging with regular expressions: use Python (or Perl)
- Robustly engineered code (even more critical with multiple programmers): use Java
- Extreme high performance with manually done memory management (painful): use C/C++

The ultimate advantage of compiling languages



source: `http://xkcd.com/303/`

Java Workflow I



Java development workflow.
source: http://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html

Java Workflow II

Workflow

- Quellcode: `.java`-Dateien
- Compiler generiert `.class`-Dateien $\rightarrow$ *Bytecode*
- Bytecode ist kein nativer, plattformabhängiger Code
- Bytecode: Maschinensprache der *Java Virtual Machine (JVM)*
- `java` führt bytecode mittels der JVM auf dem Endgerät aus

Vorteile und Eigenschaften

- Kontrollierte virtuelle Maschine
- JVM für alle gängigen Plattformen verfügbar
- JVM bietet erweiterte Funktionalität, z.B. Garbage Collector (GC)
- Optimierung der JVM für bestimmte Prozessoren/Architekturen
- Skalierung!

Plattformunabhängigkeit und Objektorientierung

Plattformunabhängigkeit

- Compiler generiert Bytecode gegen JVM
- Abstraktion von zentralen Betriebssystem-Schnittstellen
- Ein-/Ausgabe, grafische Benutzerschnittstellen etc. mit gemeinsamer API
- Entwickler: Programmierung gegen Java API

Objektorientierung

- Ziel: Schreiben großer, fehlerfreier Anwendungen
- Annahme: Menschen denken objektorientiert
- Modellierung von Verbindungen und Kommunikation von Objekten in einer Programmiersprache
- Java weitgehend aber nicht vollständig objektorientiert. Ausnahme: primitive Datentypen

Why Java? I

- Betriebswirtschaftliche Faktoren: weite Verbreitung
- Hoher Stellenwert im universitären Umfeld
- Hoher Bedarf an Java-Entwicklern
- Verfügbarkeit vieler Bibliotheken (auch: NLP)
- Einführung in viele Konzepte moderner Programmierparadigmen

Zahlen [a]

[a] source: http://www.oracle.com/us/corporate/press/193190

- 1.1 Milliarden Desktops mit Java
- 930 Millionen Downloads der JRE pro Jahr
- 100% Installationsbasis auf Blu-ray Playern
- 500 Millionen aktivierte Android-Geräte (Stand: September 2012)

Why Java? II

Geschwindigkeit

- Just-In-Time (JIT)-Compiler
- Übersetzung von relevanten Codeteilen während der Laufzeit
- Geschwindigkeit in vielen Fällen vergleichbar mit kompilierten Programmiersprachen

Speicherverwaltung

- Keine Pointer, sondern Referenzen für Verweise auf Variablen
- Typ der Referenz nicht änderbar
- Definierbare Sichtbarkeit von Variablen
- Garbage Collector: automatisches Löschen von Objekten und Speicherfreigabe bei Bedarf

Why Java? III

Syntax

- Simple explicit syntax makes Java code easy to read and understand
- Disadvantage: "verbosity" but the IDE (for example, Eclipse) can help with this

Software Engineering

- Java enforces good habits in software engineering
- Object-oriented view forces programmers to think ahead more (much more on this later!)
- Handling possible errors (through "Exceptions") is required
- Static type checking reduces errors (i.e., if you try to assign a string to an integer, or compare a string and an integer, compilation error)
- Disadvantage, again: "verbosity"

Paradigmen der Programmierung

Vielzahl von Kriterien zur Unterscheidung von Programmiersprachen:

- prozedural vs. funktional vs. deklarativ vs. objekt-orientiert
- low-level vs. high-level
- kompiliert vs. interpretiert vs. bytecode-kompiliert

Paradigmen der Programmierung

Vielzahl von Kriterien zur Unterscheidung von Programmiersprachen:

- prozedural vs. funktional vs. deklarativ vs. objekt-orientiert
- low-level vs. high-level
- kompiliert vs. interpretiert vs. bytecode-kompiliert

Python vs. Java

- Python: interpretiert und prozedural (größtenteils)
- Java: bytecode-kompiliert und objekt-orientiert

Java Versions & IDE I

JDK, JRE, JVM

- Zur Entwicklung: Java Platform Standard Edition (Java SE)
- Java Development Kit: Compiler, JVM und Java Bibliotheken
- Download: `http://www.oracle.com/technetwork/java/javase/downloads/index.html`
- Alternative: OpenJDK (Referenzimplementierung)
- Aktuelle Java-Version: Java 8 (seit 2013)

Java Versions & IDE II

Interesting Java Versions

- Java 1.3: used in Java ME (e.g., games on old cell phones)
- Java 1.5 (also called Java 5): heavily used, examples in most of our books. Beginning of "modern Java" in my view.
- Java 1.7 (also called Java 7): added better file manipulation (but old ways still work!)
- Java 1.8 (also called Java 8): adds some features that are not yet widely used
- VERSION FOR THIS COURSE: Java 7 (but Java 8 is also fine)

Goals of this session

- ☑ Details, getting a Schein
- ☑ Introduction to Java, why learn Java?
- ☐ How to write a “hello world” program in java

Writing our first Java program

Prerequisites

- JDK (JRE is not enough!)
- Development Environment:
  - Either: command line + text editor (e.g. `vim`)
  - Or: IDE – integrated development environment (eclipse, netbeans, ...)
- Recommendation for this course: **eclipse**
- This session: command line + editor

Hello World!

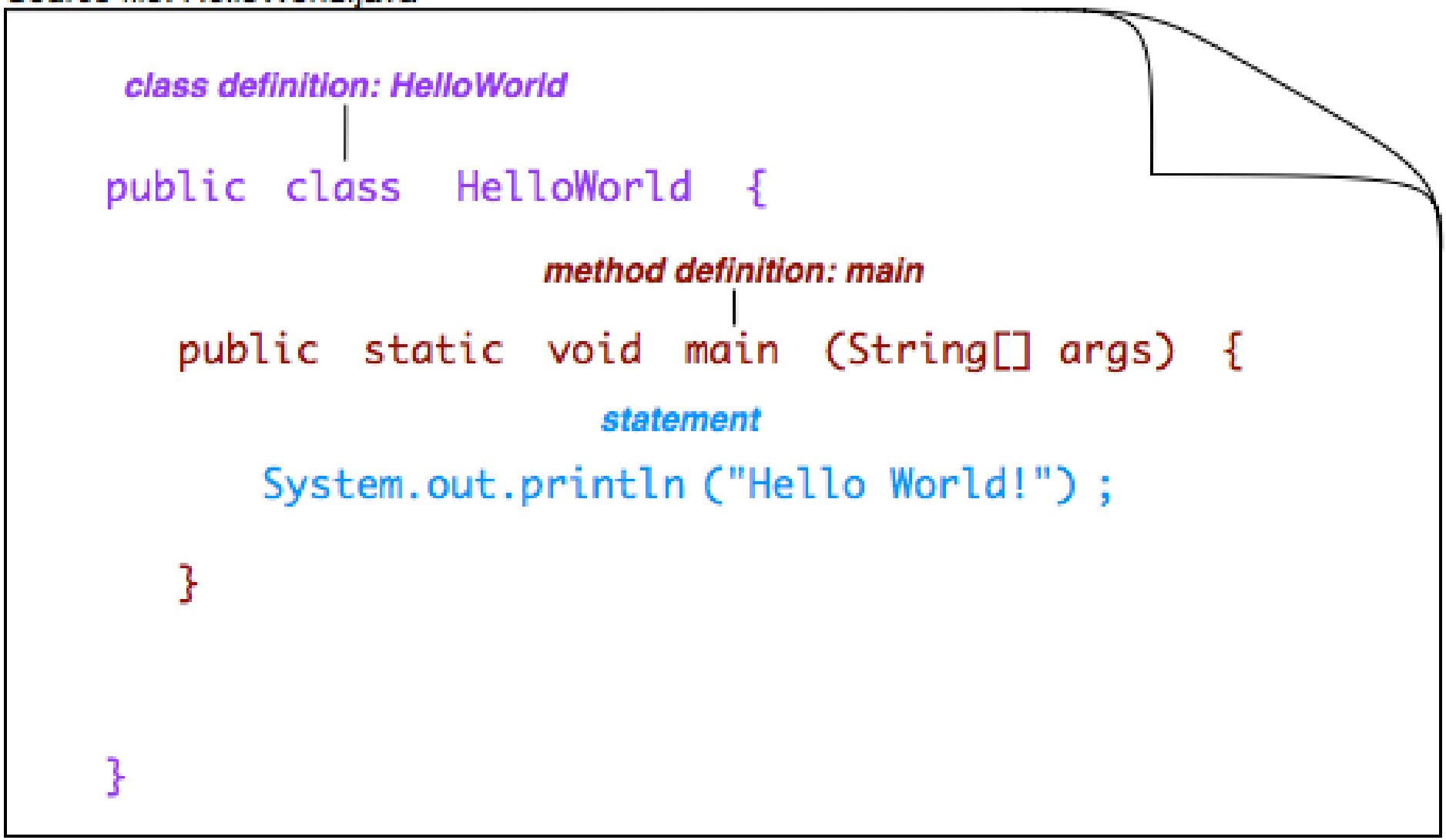
```java
// file name == class name
public class HelloWorld {
    // a method (function) called "main"
    // parameter input: array of strings
    public static void main( String[] args ) {
     System.out.println( "Hello world!" );
    }
}
```

code/HelloWorld.java

1. Save
2. Compile: `javac HelloWorld.java` → `HelloWorld.class`
3. Run: `java HelloWorld`

Anatomy of a class I

Source file: HelloWorld.java

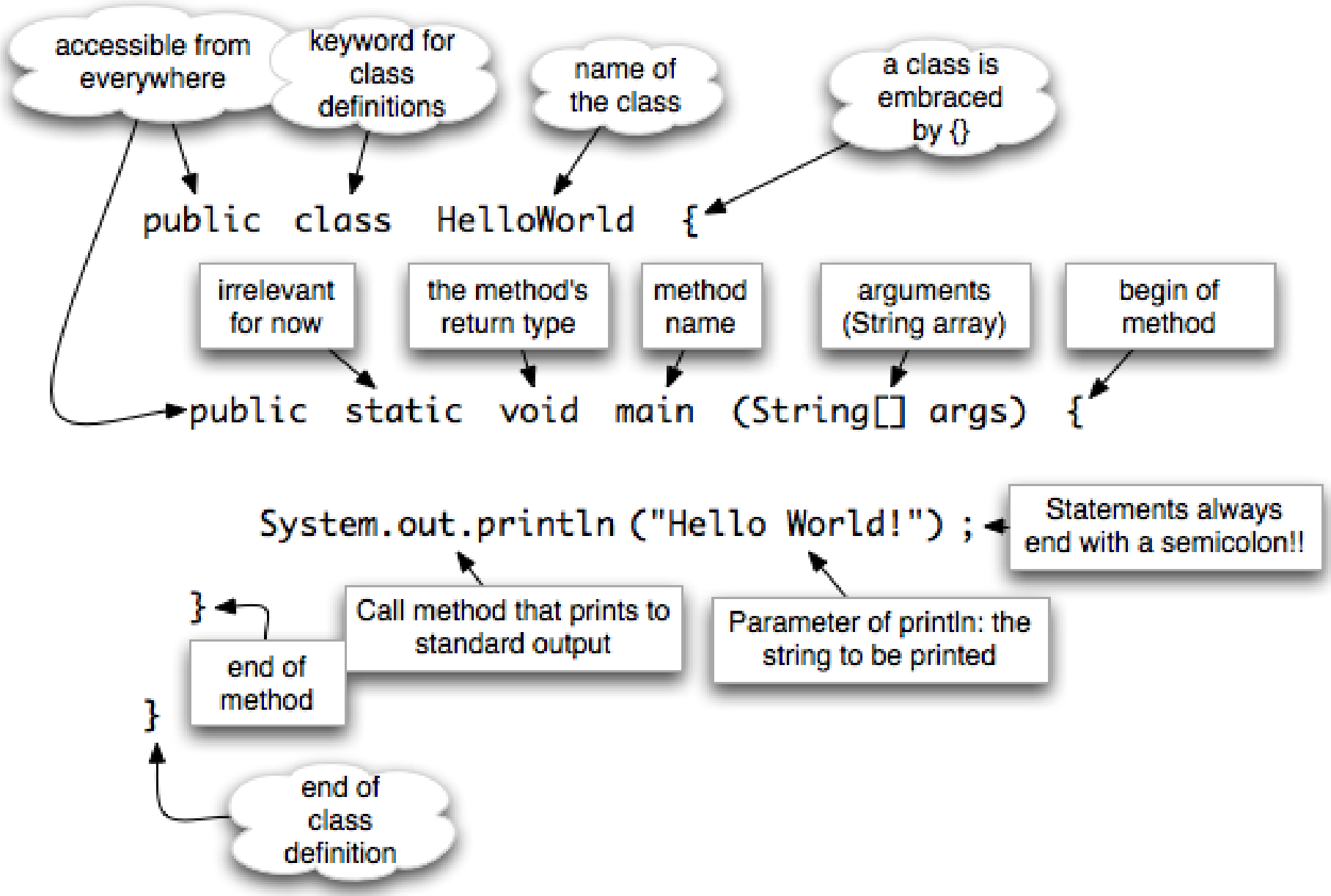


Anatomy of a class II

Layered structure (simplified)

- source file (`.java`): *one class definition*
- class definition: one or more *methods*
- method: one or more *statements*

Hello World explained



Comments? Comments!

Comments in one line

- Comment in one line: `//`

```
// this is a comment
```

Comments spanning multiple lines

- Comments spanning multiple lines: `/* */`

```
/* this is a comment
   this is still a comment  */
```

Javadoc comments

- Javadoc comments: `/** */`

```
/** javadoc comment
    comment continued   */
```

Experiments

Testing the compiler

- What happens if the class name differs from the file name?
- Do we need a main method?
- Do we need to specify `void`?
- Is it possible to print numbers instead of strings?
- ...

Goals of this session

- ☑ Details, getting a Schein
- ☑ Introduction to Java, why learn Java?
- ☑ How to write a “hello world” program in java

Preparation for the next session

- Install the JDK, Java version 7
- Adapt your environment variables if necessary
- Compile and run our `HelloWorld` class
- Experiment with the code: make errors and check how the compiler behaves
- Recommendation: install eclipse (`http://www.eclipse.org/downloads/`)
- (optional) bring your notebook to the lecture

General Java Literature

The official Java Tutorial

`http://docs.oracle.com/javase/tutorial/getStarted/index.html`

Eckel, B.

Thinking in Java.
Prentice Hall, 2006.

Sierra, K. & Bates, B.

Head First Java.
O'Reilly Media, 2005.

Ullenboom, Ch.

Java ist auch eine Insel.
Galileo Computing, 2012.
NOTE: see the HTML index!

Java Style

Java Style Guides

- Java Ranch is widely recommended:
  http://www.javaranch.com/style.jsp (click on the categories!)
- The Google Java style guide is also interesting:
  http://google-styleguide.googlecode.com/svn/trunk/javaguide.html

Special Topics

Design Patterns

Gamma, E. et al.

Design Patterns. Elements of Reusable Object-Oriented Software.
Addison-Wesley Longman, 2008.

Test-Driven Development

Beck, K.

Test-driven development – by example.
Addison-Wesley, 2005.

Android Development

- Android official (see “first app”):
  http://developer.android.com/training/index.html
- Recommended:
  http://www.makeuseof.com/tag/write-google-Android-application/