ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Java-Einführungskurs

Informatik II für D-ITET, FS 2013, ETH Zürich

Simon Mayer
simon.mayer@inf.ethz.ch

Ziele / Überblick

- Vorbereitung auf die Übungen zu Informatik II

- Theorie
  - Die Java-Technologie und Sprache
  - Unterschiede zwischen C++ und Java

- Praxis: Übungsblatt 0
  - HelloWorld.java
  - Erste Schritte mit Eclipse
  - JUnit 4 für automatisiertes Testen

Warum Java (Marketing)

- Objektorientiert
- Einfacher als C++
- Virtuelle Maschine: Compile once – Run everywhere
- Umfangreiches Ökosystem: Tools, Bibliotheken, ...

Die Java-Technologie

- Java-Entwicklungswerkzeug: JDK
  - Enthält die Programme `java`, `javac`, `javap...`
- Java-Laufzeitumgebung: JRE
  - Hauptbestandteil ist das Programm `java`
  - Eine Art „Virtueller Computer“
  - Java Virtual Machine (JVM)
  - Standardklassen und weitere Programmbibliotheken
- JRE-Editionen: Java SE, Java ME, Java EE

Werdegang eines Java-Programms

Program.java — **Quellcode**: Menschenverständlicher Text

↓ `javac Program.java` — Java-Compiler

Program.class — **Java-Bytecode**: Maschinenverständlicher Code

↓ `java Program` — Virtuelle Maschine

Plattformunabhängigkeit: Bytecode ist ohne Änderung auf jeder Architektur lauffähig, auf welcher eine Laufzeitumgebung installiert ist :-D



Downside??

Das Java-Ökosystem

Java-Ökosystem

- Standardbibliothek
  - Datenstrukturen (List, Map,...), Algorithmen (Sort,...), Kryptographie, Kommunikation, graphische Benutzerschnittstellen
- Zusätzliche Bibliotheken
  - Datenbanken, Web-Server,...
- Integrierte Entwicklungsumgebung (IDE)
  - Editor + Compiler + Debugger + Projektverwaltung + ...
  - Beispiel: **Eclipse**, NetBeans, IntelliJIDEA, ...

Java-Ökosystem

- Javadoc
  - Dokumentation durch *strukturierte* Kommentare
- Unit Testing (JUnit)
  - Bestandteil aller Übungen
  - Automatisiertes Testen des Codes
  - Generierung von Testberichten

Die Java-Sprache

Demo: HelloMachine.java

```
public class HelloMachine {

    public static void main(String [] args) {
        String userName = args[0];

        HelloMachine myHelloMachine = new HelloMachine();

        myHelloMachine.sayHello(userName);
    }

    public void sayHello(String name) {
        System.out.println("Hello, " + name + "!");
    }
}
```

Beachten: Funktionssignaturen, besonders keyword „static“; Parameter; System.out.println(...); Objekte;

Java-Sprache: Versionen

- JDK 1.0 (1996)
- JDK 1.1 (1997, z.B. Paketierung als .jar-Dateien)
- J2SE 1.2 (1998, z.B. Just-In-Time Compiler, Grafik)
- J2SE 1.3 (2000)
- J2SE 1.4 (2002, z.B. + Assertions und Server)
- Java 5.0 (2004, z.B. + Generics, Annotationen)
- Java 6.0 (2006)
- **Java 7.0** (2011, z.B. neue Filesystem-API, IPv6)
- Java 8.0 (2013)

Java-Sprache: Organisation

- Pro öffentliche Klasse / Schnittstelle eine *gleichnamige .java*-Datei
- Pro (öffentlicher oder nicht-öffentlicher) Klasse generiert javac eine *.class*-Datei

Java-Sprache: Pakete

- Klassen können Teil eines „Pakets“ sein
  - Definition: `package myPackage;`
  - Navigation: `.`
- Ziele
  - Vermeidung von Namenskollisionen
  - Kompakterer, einfacher lesbarer Code
- Pakethierarchie wird auf Verzeichnisbaum abgebildet!

Java-Sprache: Bibliotheken

- Sammeln von Paketen in .jar-Dateien (java archive)
- Zugriff auf Bibliothek
  - Bekanntmachen des Namens: `import` ...
  - Namen aus dem eigenen Paket sind immer bekannt
- Standardbibliothek steht automatisch zur Verfügung: http://java.sun.com/javase/7/docs/api/
  - Kein import von z.B. `java.lang` nötig

Demo: Organisation, Pakete, Zugriff

- `Public.java`
  - Klasse „Public“ in Package „demo1“
  - `public static void main() { … }`
  - Benutzt Klasse ExtendedPublic

- `ExtendedPublic.java`
  - Klasse „ExtendedPublic“ und private Klasse „Private“ in Package „demo2“
  - `public void foo() { … }`

Java-Sprache: Zugriffsrechte

- `private`
- `public`
- `protected`
- `package`

- Keine `friends`

Java-Sprache: Objekte

- Objekt: Instanz einer Klasse
- Zugriff ausschliesslich über Referenzen!
- Erzeugung mit `new`
- Entfernung durch Garbage Collector, kein `delete`

Java-Sprache: Primitive Typen

- Primitive Typen können auf dem Stack angelegt werden, ihre Instanzen sind **keine Objekte**!

- `boolean`
- `byte, short, int, long`
- `float, double`
- `char`

- Korrespondierende Klassen, z.B. `int vs. Integer`
  - Werden, wie alle Objekte, als Referenzen by Value übergeben (mehr dazu in Übung 3) und am Heap allokiert

Java-Sprache: Vererbung

- Java bietet keine Mehrfachvererbung an
  - Stattdessen: Schnittstellen (`interface`)
  - Eine Klasse kann mehrere Interfaces implementieren
  - Mehr dazu in Übung 6

- Polymorphismus: Funktionen sind grundsätzlich virtuell!
  - Wenn: `Car extends Vehicle`
  - Dann: `Car.getSpeed()` überdeckt `Vehicle.getSpeed()`

- Funktionen und Klassen können `abstract` sein
  - Abstrakte Klassen können nicht instanziiert werden

Java-Sprache: Fehler und Stack Traces

- Stack Traces ermöglichen das Zurückverfolgen von Fehlern zu ihrem Ursprung (+ Zeilennummern)
- Siehe Demo...

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at StackTraceDemo.method2(StackTraceDemo.java:20)
    at StackTraceDemo.method1(StackTraceDemo.java:12)
    at StackTraceDemo.main(StackTraceDemo.java:7)
```

Java-Sprache: Fehlerbehandlung

- Ein **Error** "indicates serious problems that a reasonable application should not try to catch."
  - Beispiel: `OutOfMemoryError`
- Eine **Exception** "indicates conditions that a reasonable application might want to catch."
  - Beispiel: `FileNotFoundException`
- Abfangen mit `try/catch`, werfen mit `throw`
  - Nicht abgefangene Exceptions (und Errors) führen zum Programmabbruch!
  - Mehr dazu in Übung 1 und 2

Java-Sprache: Weitere Features

- Annotationen
  - Einbindung von Metadaten

```
public class Vehicle {
    double getSpeed() {}
}

public class Car extends Vehicle {
    @Override
    double getSpeed() {}
}
```

- Generics (siehe Übung 7)

```
public class Parkhaus {
    List<Vehicle> getVehicles() {}
}
```

- Reflection: Introspektion eines Programms
  - Programm kennt seine eigene Struktur

Übungsblatt 0

Übungsblatt 0: Aufgabe 1

- HelloWorld mit Texteditor

- HelloWorld in Eclipse
  - Falls schon installiert: Super!
  - Sonst: Links zu Dokumentation auf Vorlesungswebsite

Übungsblatt 0: Aufgabe 1 und Eclipse

- Import der Übungsdaten in Eclipse
  - File – New - Java Project – [Projektnamen Eingeben]
  - Next – Link Additional Source – Übungsordner auswählen (z.B. „u0“) - Finish
- Alternativ: Übungsordner in neues Projekt schieben
- Neue Bibliothek einbinden
  - Rechtsklick auf Projektname – Build Path – Configure Build Path – Reiter „Libraries“
  - „Add (external) JARs“ - Einbinden von JAR-Dateien
  - „Add Library“ - Einbinden von z.B. JUnit

Übungsblatt 0: Aufgabe 2

- Erstes Java-Programm: Signum-Funktion

Übungsblatt 0: Aufgabe 3

- Automatisiertes Testen mit JUnit4
- … aus der Kommandozeile
- … in Eclipse

Übungsblatt 0: Aufgabe 4

- Modellbildung, für zu Hause...

Noch Fragen :-)