Erster Kontakt mit Java und Pseudocode

CoMa-Übung II

TU Berlin

23.10.2013

Themen der Übung

1. Java auf meinem Rechner
2. Hello World
3. Pseudocode
4. Einige Datentypen
5. Modulo-Rechnen
6. Bedingte Anweisungen
7. Vergleiche und logische Operatoren
8. Fallunterscheidungen

Java auf meinem Rechner

Installation

- Neueste Version des Java JDK herunterladen (`http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html`)
- JDK = Java Development Kit, JRE = Java Runtime Environment
- JDK enthält alles Nötige um Java-Dateien zu entwickeln und auszuführen
- Es gibt Installationsanweisungen für jedes Betriebssystem (`http://docs.oracle.com/javase/7/docs/webnotes/install/`)
- Merkt euch, wohin ihr das JDK installiert!

Java-Programme schreiben

Vom Programmierer zum Programm

- Programmierer nimmt einen beliebigen Text-Editor und legt eine neue `.java`-Datei an (z.B. `HelloWorld.java`)
- Ist die Datei fertig, kann die Text-Datei mit `javac HelloWorld.java` (Kommandozeile) in Java-Bytecode übersetzt werden
- Java-Bytecode kann auf allen Systemen ausgeführt werden, für die es eine Java Virtual Machine (JVM) gibt (`java HelloWorld`)
- JVM ist Teil des JREs, der Compiler Teil des JDKs

Java – Konzepte

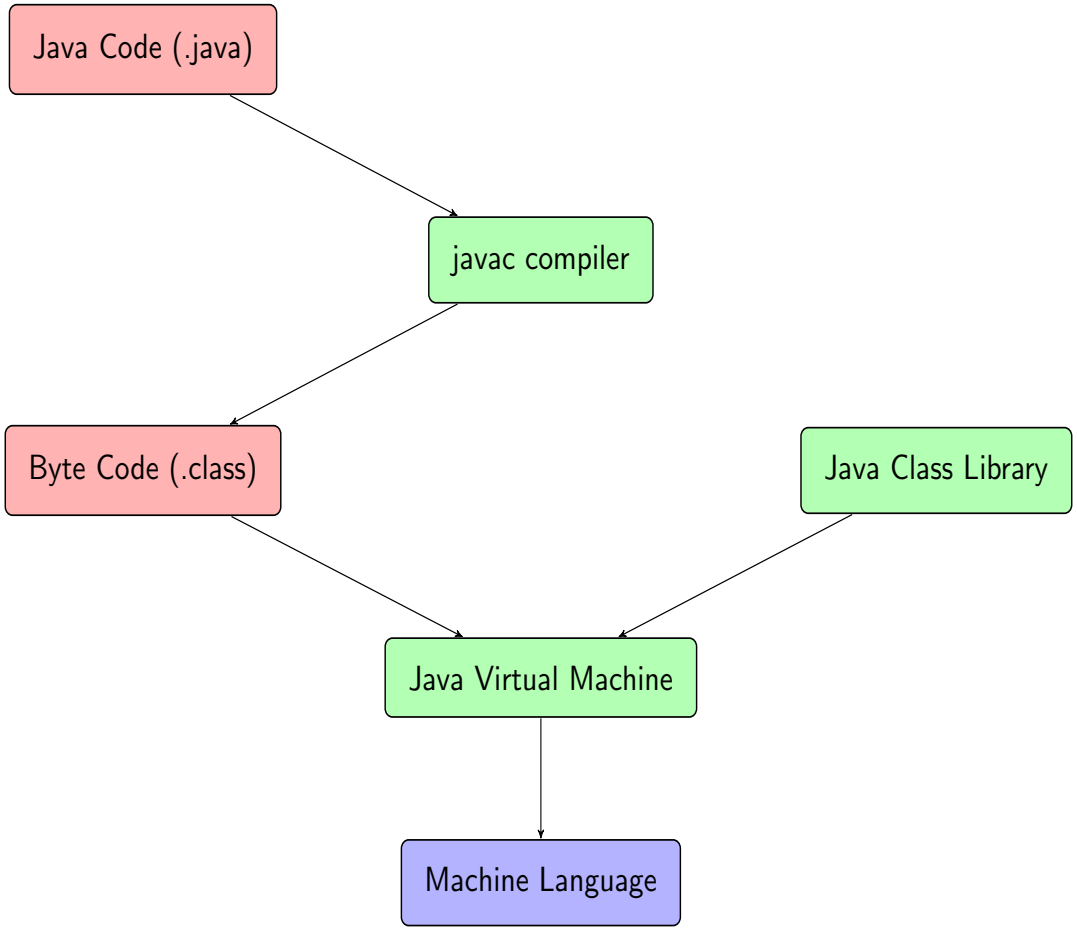
Konzepte

- Java ist eine objekt-orientierte Programmiersprache
- Man fängt mit sehr einfachen Datentypen (z.B. Zahlen) an und baut daraus komplexere Strukturen (Klassen/Objekte)
- Aus diesen Strukturen baut man dann noch komplexere Strukturen, etc.

Klassenbibliotheken

- Wenn jeder bei Null anfangen würde, wäre das sehr ineffizient
- Es gibt große Bibliotheken voller vorgefertigter Strukturen (die ihr auch als Teil des JDKs bekommen habt)
- Werden wir nach und nach einführen

Ausführung einer java-Datei



Hello World

Test

- Erstellung der Datei `HelloWorld.java`
  - Groß- und Kleinschreibung beachten
  - Aufpassen, dass kein zusätzliches `.txt` an den Dateinamen angehängt wird
- Zum Beispiel mit den Editoren gedit, emacs, vim . . . oder den Entwicklungsumgebungen Eclipse, NetBeans . . .

Hello World

Beispielprogramm

```
1 public class HelloWorld {
2
3     public static void main(String[] args) {
4         System.out.println("Hello World!");
5     }
6
7 }
```

- Die Klasse muss den gleichen Namen haben wie die Datei (bei public class).
- Der Klassenname muss mit einem Buchstaben beginnen. (Nach Konvention beginnt er mit einem Großbuchstaben.)
- Zum Ausführen wird eine sogannte *main*-Methode benötigt.

Hello World (Fortsetzung)

Test

- Start eines Terminals, Navigation in das Verzeichnis mit `HelloWorld.java`
- Kompilieren der Datei mit `javac HelloWorld.java`
- Start des Programms mit `java HelloWorld`
- Wenn alles gut geht, sollte `Hello World!` ausgegeben werden

Die Bestandteile von Hello World

```
1 public class HelloWorld {
2
3     public static void main(String[] args) {
4         System.out.println("Hello World!");
5     }
6
7 }
```

class

- Sagt, dass ihr jetzt eine neue Klasse (= eine der erwähnten komplexeren Strukturen) definieren wollt

Sichtbarkeitsmodifikatoren: `public`, `private`

- Schlüsselwörter wie `public` und `private` bestimmen, welche Teile eurer Klassen von außen sichtbar sind
- Bsp.: In einem Handy wäre das Display `public` (da von außen sichtbar), der Prozessor aber nicht.

Die Bestandteile von Hello World

```java
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello World!");
    }

}
```

`public static void main(String[] args)`

- Definiert die Einstiegsmethode für ein Programm, damit Java weiß wo das Programm anfängt

`System.out.println()`

- Ruft die Methode `println()` der Variable `out` der Klasse `System` auf
- Gibt etwas in der Kommandozeile aus

Hello World – Zusammenfassung

```java
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello World!");
    }

}
```

Was ihr von dem Programm mitnehmen solltet:

- `System.out.println(x)` gibt `x` auf der Kommandozeile aus und beginnt eine neue Zeile
- `x` kann dabei eine Zahl, eine Zeichenkette oder ein beliebiges komplexeres Objekt sein
- Ihr kennt jetzt ein Grundgerüst, um ein ausführbares Java-Programm zu erstellen

Pseudocode

Probleme

- Fließtext-Beschreibungen von Algorithmen: meist lang & nicht präzise
- Java-Code: schwerer zu verstehen, viele technische Details
- $\rightarrow$ Pseudocode als Mittelweg

Idee von Pseudocode

- Code für Menschen statt für Computer
- Relativ präzise und exakt, aber keine technischen Details
- Keine sprach-spezifische Syntax
- Keine expliziten Typdeklarationen
- Keine Effizienz-Tricks
- $\rightarrow$ wird daher oft in der Literatur verwendet

Pseudocode – Grundform & Beispiel

Grundform

- `Algorithmenname(Parameterliste)`
- `Input:`
- `Output:`
- `Liste der Programmschritte`

Beispiel

```
maximum(a,b)
Input:   a, b ∈ ℝ
Output:   max{a, b}
IF (a > b) THEN
     RETURN a
ELSE
     RETURN b
ENDIF
```

Pseudocode – Befehle

Bedeutung	Pseudocode	Java-Code
Zuweisung	:=	=
Vergleich	=, ≠, ≤, ≥, <, >	==, !=, <=, >=, <, >
Logisches Und	AND, ∧	&&
Logisches Oder	OR, ∨	||
Logisches Nicht	NOT, ¬	!
Kommentar	//	//, /* */
Rückgabe	RETURN	return

Bedingte Anweisungen

```
IF (condition) THEN
      ...
ELSE IF (condition2) THEN
      ...
ELSE
      ...
ENDIF
```

```
1 if (condition) {
2
3 } else if (condition2) {
4
5 } else {
6
7 }
```

Pseudocode – Befehle (2)

while-Schleifen

```
WHILE (condition) DO
    ...
ENDWHILE
```

```
DO
    ...
WHILE (condition)
```

```java
while (condition) {

}
```

```java
do {

} while (condition);
```

for-Schleifen

```
FOR i := 1 TO n DO
    ...
ENDFOR
```

```
FORALL a ∈ A DO
    ...
ENDFOR
```

```java
for (int i=1; i<=n; i=i+1) {

}
```

```java
for (Datentyp a : A) {

}
```

Pseudocode – Zusammenfassung

Einfache Algorithmen

- Einfache Operationen / Algorithmen sind in Pseudocode erlaubt
- `A := {b,a,c}`
- `sortiere A`
- b := `wähle ein zufälliges` $a \in A$

Zusammenfassung

- HA fragt nach Java: kein Pseudocode
- HA fragt nach Algo: Pseudocode
- Keine Algorithmen als Fließtext!
- Kopf des Pseudocode (Name, Parameter, Input, Output) wichtig
- Kein `goto` erlaubt
- `.`-Operator erlaubt

Einige Datentypen

Zahlen

- Zur Zahldarstellung stehen in Java unter Anderem die Typen `int` und `double` zur Verfügung
- `int` entspricht hierbei ganzen Zahlen, `double` den reellen Zahlen
- Eine Variable vom Typ `boolean` kann den Wert `true` oder `false` haben, also einen Wahrheitswert.

Achtung

- Rechnung mit `int` weist einige Besonderheiten auf
- In Java werden `double`-Zahlen mit "." und nicht mit "," geschrieben.

Einige Datentypen

Zeichenketten in Java

- Die Klasse `String` stellt nicht-veränderbare Zeichenketten dar
- Zeichen-Konstanten werden durch doppelte Anführungszeichen ausgewiesen: z.B. `"Hello"`
- Der Konkatenations-Operator + hängt zwei Strings aneinander `"Hello" + "World"` und erzeugt einen neuen String `"HelloWorld"`
- Bekommt der Konkatenations-Operator einen String und etwas anderes, wird das andere in eine String-Darstellung umgewandelt

Nützliche Methoden der Klasse `String`

- `s.length()`: gibt die Anzahl an Zeichen in `s` zurück
- `s.charAt(int i)`: gibt das Zeichen an Position `i` in `s` zurück. Das erste Zeichen ist bei 0, das letzte bei `s.length()-1`.
- `s.equals(String t)`: sind `s` und `t` gleich? (`==`, `!=` funktionieren hier nicht, da wir es nicht mit primitiven Datentypen zu tun haben).

Modulo-Rechnung

Division mit Rest

Sei $a \in \mathbb{Z}$ und $b \in \mathbb{N} \setminus \{0\}$. Dann gibt es eindeutig bestimmte Zahlen $k, r \in \mathbb{Z}$, so dass gilt

$$a = k \cdot b + r, \quad r \in \{0, 1, \ldots, b-1\}$$

r wird als der **Rest** bezeichnet, der bei der Division von a durch b entsteht.

Schreibweisen

Wir bezeichnen r auch als

$$a \mod b \qquad \text{(lies: } a \textbf{ modulo } b\text{)}$$

Außerdem schreiben wir

$$a \equiv b \mod n \qquad \text{(lies: } a \textbf{ kongruent } b \textbf{ modulo } n\text{)}$$

wenn $a \mod n = b \mod n$, d.h. wenn a und b den gleichen Rest bei Division durch n haben.

Modulo-Rechnung in Java

Division mit Rest in Java

Den Rest einer Division kann man in Java mit dem Operator % erhalten. Er ist für Zahlen a, b wie folgt definiert:

$$\texttt{a \% b = a - (int)(a/b)} \cdot \texttt{b;}$$

- Der % Operator funktioniert auch für Gleitkommazahlen
- Das `(int)` sorgt dafür, dass aus `a/b` in jedem Fall ein `int` wird (notfalls wird abgerundet)
- Bei dieser Definition kann ein negatives Ergebnis herauskommen!
- Anders als bei mathematischer Definition, Vorsicht!

Beispiele

- $42 \mod 5 = (8 \cdot 5 + 2) \mod 5 = 2 \mod 5$, d.h. $42 \equiv 2 \mod 5$
- $-47 \mod 11 = (-5 \cdot 11 + 8) \mod 11 = 8 \mod 11$, d.h. $-47 \equiv 8 \mod 11$
- $-47 \,\%\, 11 = -47 - (int)(-47/11) \cdot 11 = -47 - (-4) \cdot 11 = -47 - (-44) = -3$

Bedingte Anweisungen

Die `if`-Anweisung

- `if (condition) doA; else doB;` führt `doA;` aus, wenn `condition` erfüllt ist, und `doB;` wenn nicht
- `condition` ist vom Typ `boolean` (d.h. ein Wahrheitswert) und kann entweder `true` oder `false` sein

```
1  public class Temperature {
2    public static void main(String[] args) {
3      boolean condition = true;
4      if (condition)
5        System.out.println("Die Bedingung ist wahr!");
6      else
7        System.out.println("Die Bedingung ist falsch!");
8
9    }
10 }
```

Vergleiche und logische Operatoren

Vergleiche

- `a == b` liefert für primitive Datentypen genau dann `true`, wenn *a* **gleich** *b* ist, und sonst `false`
- `a != b` liefert für primitive Datentypen genau dann `true`, wenn *a* **nicht gleich** *b* ist, und sonst `false`
- `a < b`, `a <= b`, `a >= b`, `a > b` liefert für Zahlen genau dann `true`, wenn $a < b$, $a \leq b$, $a \geq b$, $a > b$ ist, und sonst `false`

Logische Operatoren

Es gibt in Java vier logische Operatoren, die mit einem oder zwei `boolean`-Werten arbeiten und einen neuen `boolean`-Wert zurückgeben.

- `!a`: **nicht** a
- `a && b`: a **und** b
- `a || b`: a **oder** b
- `a ^ b`: **entweder** a **oder** b (exklusives oder, xor)

Logische Operatoren

Übersicht

boolean a	boolean b	!a	a && b	a || b	a ^ b
true	true	false	true	true	false
true	false	false	false	true	true
false	true	true	false	true	true
false	false	true	false	false	false

Logischer Kurzschluss

- Java wertet Boolschen Ausdruck nur aus, bis der Wert feststeht
- `true || a`, `false && b` $\rightarrow$ die Werte von `a` und `b` sind egal
- Das wird als *logischer Kurzschluss* bezeichnet
- Soll auf jeden Fall der ganze Term ausgewertet werden, gibt es alternative Und/Oder-Operatoren `&` und `|`
- Gleicher Effekt, aber ohne logischen Kurzschluss

Operatorenvorrang (Höchste zuerst)

Operatoren	Beschreibung
`+,-,!`	Unäres Plus, unäres Minus, logisches Nicht
`*,/,%`	Multiplikation, Division, Rest
`+,-`	Addition, Subtraktion, Konkatenation von Strings
`<,>,<=,>=`	Numerische Vergleiche
`==,!=`	Gleichheit
`&`	Logisches Und
`^`	Logisches Xor
`|`	Logisches Oder
`&&`	Logisches konditionales Und
`||`	Logisches konditionales Oder
`?:`	Bedingungsoperator

- Bei gleicher Priorität wird von links nach rechts ausgewertet
- 1 Argument $>$ 2 Argumente $>$ 3 Argumente, Punkt- vor Strich
- Numerische Vergleiche $>$ Gleichheit $>$ Und $>$ Oder

Zurück zu `if`

Mehrere Anweisungen

- Sollen bei einer `if`-Abfrage mehrere Anweisungen ausgeführt werden, müssen Blöcke benutzt werden
- `if (condition) { doA; doB; } else { doC; doD; }`
- Für komplexere Fallunterscheidungen gibt es das Konstrukt
- `if (c1) { doA; } else if (c2) { doB } else { doC; }`
- Beliebig viele `else if`'s
- Der erste passende Fall wird ausgeführt, der Rest nicht

```
1  if (number == 0) {
2    sign = 0;
3  } else if (number > 0) {
4    sign = 1;
5  } else {
6    sign = -1;
7  }
```

Beispiele

```java
boolean a = true && false || true;
boolean b = !(true || false);
boolean c = !a ^ !b;
int number = 1;
if (c) {
   number = number * 2;
} else if (c && number > 0) {
   number = number * 3;
} else {
   number = 0;
}
```

Was kommt raus?

- `true && false || true` → `false || true` → `true`
- `!(true || false)` → `!(true)` → `false`
- `!a ^ !b` → `!true ^ !false` → `false ^ !false` → `false ^ true` → `true`
- `number = 2`