

To know recursion,

you must first know recursion.

Rekursion: Beispiele

- Bier trinken
- 8-Damen-Problem
- iPod Shuffle für alle Mitarbeiter
- Karten sortieren mit MergeSort

Rekursive Methoden

// Methoden, die sich selbst aufrufen

// indirekt rekursiv

$f() \rightarrow g() \rightarrow f()$

// direkt rekursiv

$f() \rightarrow f()$

Rekursion allgemein

```
if (Problem klein genug)
    nicht rekursiver Zweig // Terminierungsfall
else
    rekursiver Zweig mit kleinerem Problem;
```

Das Problem wird bei jedem Aufruf signifikant “kleiner”, d.h., der Abstand zum Terminierungsfall wird kleiner.

Zahlen iterativ summieren


$$\text{sum}(n) = 0 + 1 + 2 + 3 + \dots + n$$

```
static long summiereterativ (long n) {  
    int summe = 0; int zahl = 1;  
    while (zahl <= n) {  
        summe = summe + zahl;  
        zahl ++;  
    } // eine for-Schleife geht auch :-)  
    return summe;  
}
```

Zahlen rekursiv summieren

$\text{sum}(n) = n + \text{sum}(n - 1)$
Terminierungsfall: $n == 0$

```
static long summiereRekursiv (long n) {  
    if (n == 0) // Terminierungsfall  
        return 0;  
    else  
        return n + summiereRekursiv(n - 1);  
}
```




***Ohne Terminierungsfall terminiert eine rekursive
Methode niemals!!!***

Zahlen rekursiv summieren

$\text{sum}(n) = n + \text{sum}(n - 1)$
Terminierungsfall: $n == 0$

```
static long summiereRekursiv (long n) {  
    if (n == 0) // Terminierungsfall  
        return 0;  
    else  
        return n + summiereRekursiv(n - 1);  
}
```

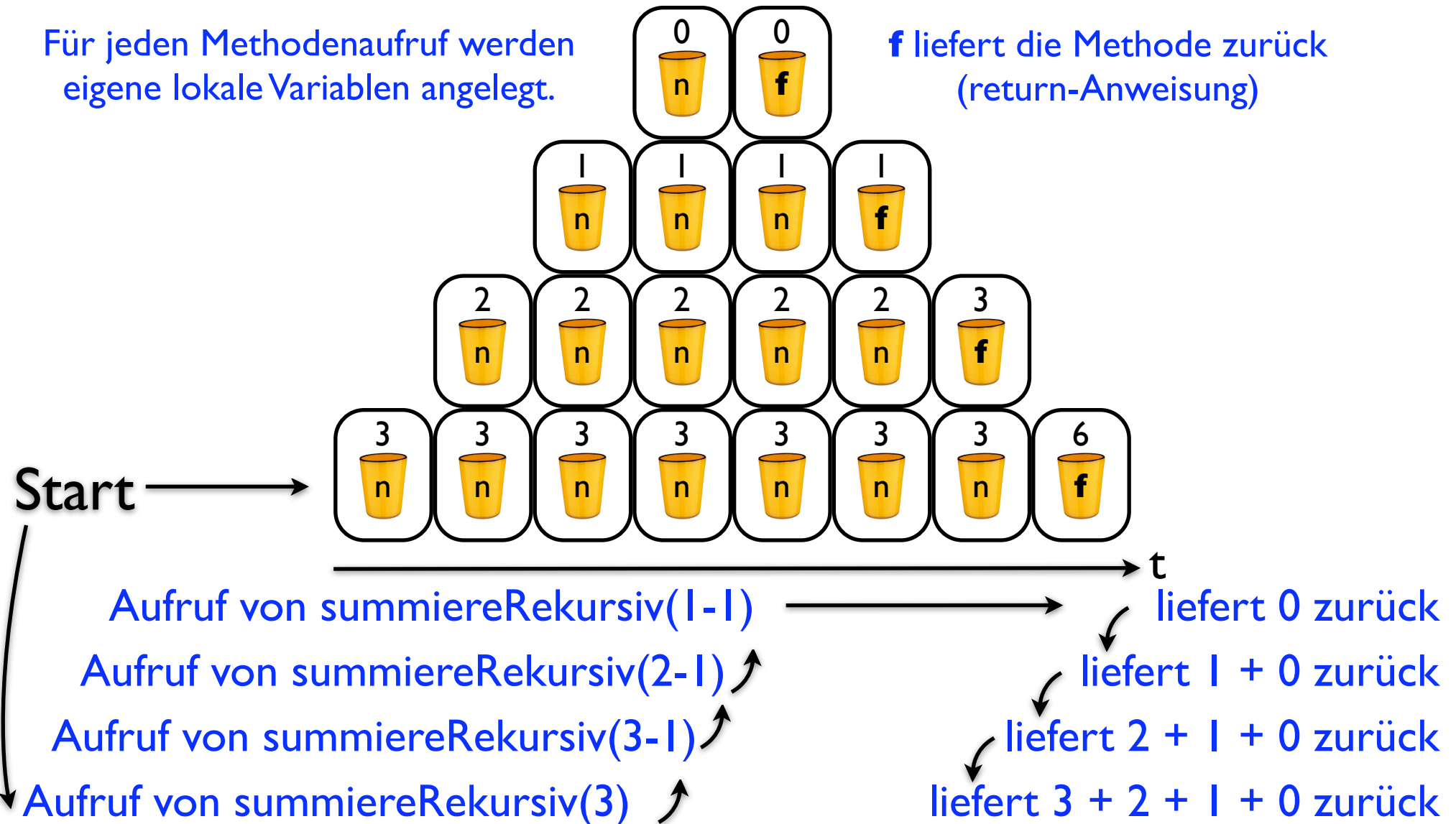


*Es gibt nur eine lokale Variable n.
Wo werden die vielen n_i gespeichert?*

Stackspeicher für lokale Variablen

Für jeden Methodenaufruf werden eigene lokale Variablen angelegt.

f liefert die Methode zurück (return-Anweisung)



Fakultät rekursiv berechnen



$$n! = n * (n-1) * (n-2) * \dots * 1$$

$$0! = 1$$

Terminierungsfall: $n == 0$

```
static public long fakultaetRekursiv (long n) {  
    if (n == 0) // Terminierungsfall  
        return 1;  
    else  
        return n * fakultaetRekursiv(n - 1);  
}
```

Rekursion vs. Iteration

Jede *rekursive Lösung* läßt sich auch *iterativ* mit Schleifen lösen.

Die *rekursive Lösung* ist oft eleganter und kürzer als die *iterative Lösung*.

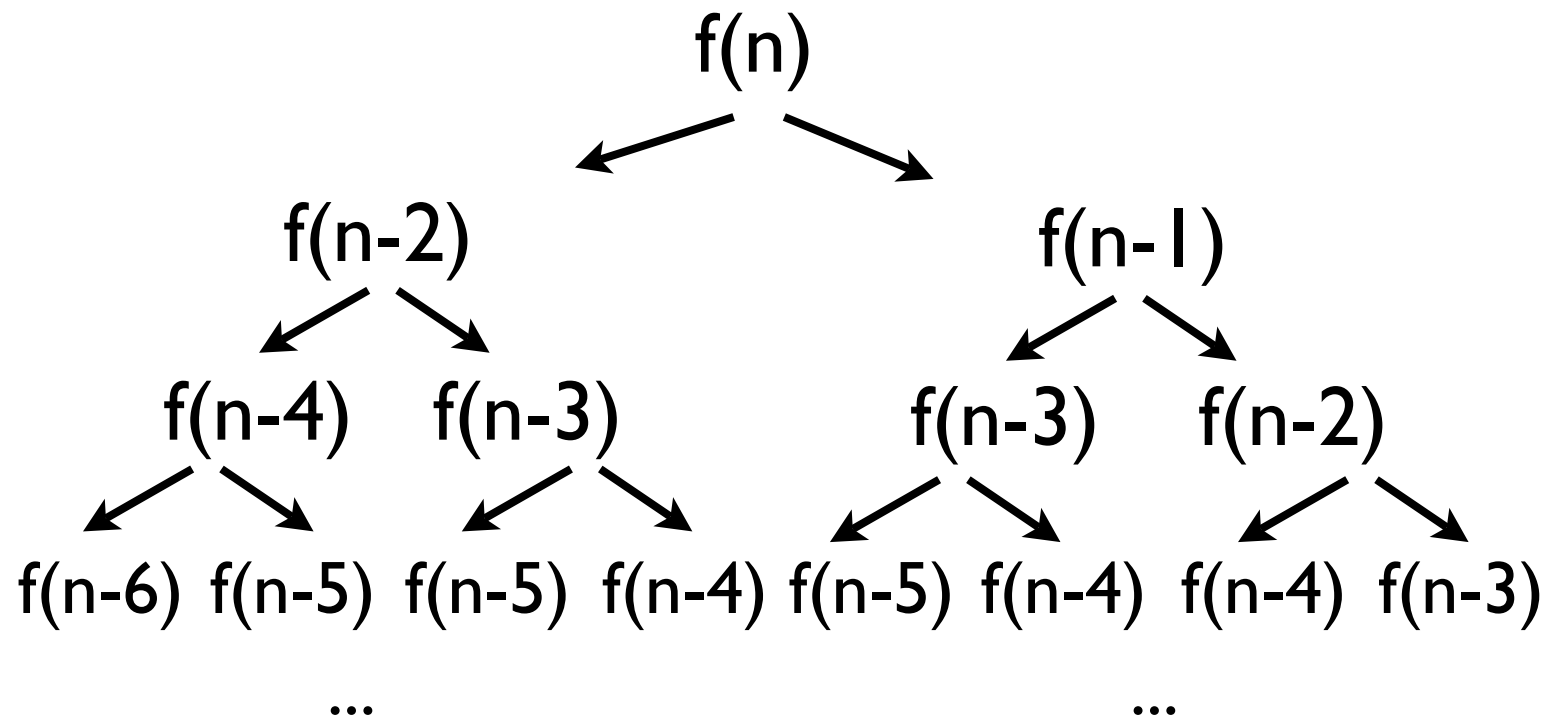
Rekursion verbraucht aber mehr Speicher und ist langsamer (lokale Variablen, Rücksprungadressen auf dem Stack).

Beispiel: Fibonacci-Zahlen

$$\text{fib}(0) = \text{fib}(1) = 1;$$

$$\text{fib}(n) = \text{fib}(n - 1) + \text{fib}(n - 2) \text{ für } n > 1$$

resultierende Folge: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...



Beispiel: Fibonacci-Zahlen



$\text{fib}(0) = \text{fib}(1) = 1$

$\text{fib}(n) = \text{fib}(n - 1) + \text{fib}(n - 2)$ für $n > 1$

// naive rekursive Implementierung mit 2^n Aufrufen!

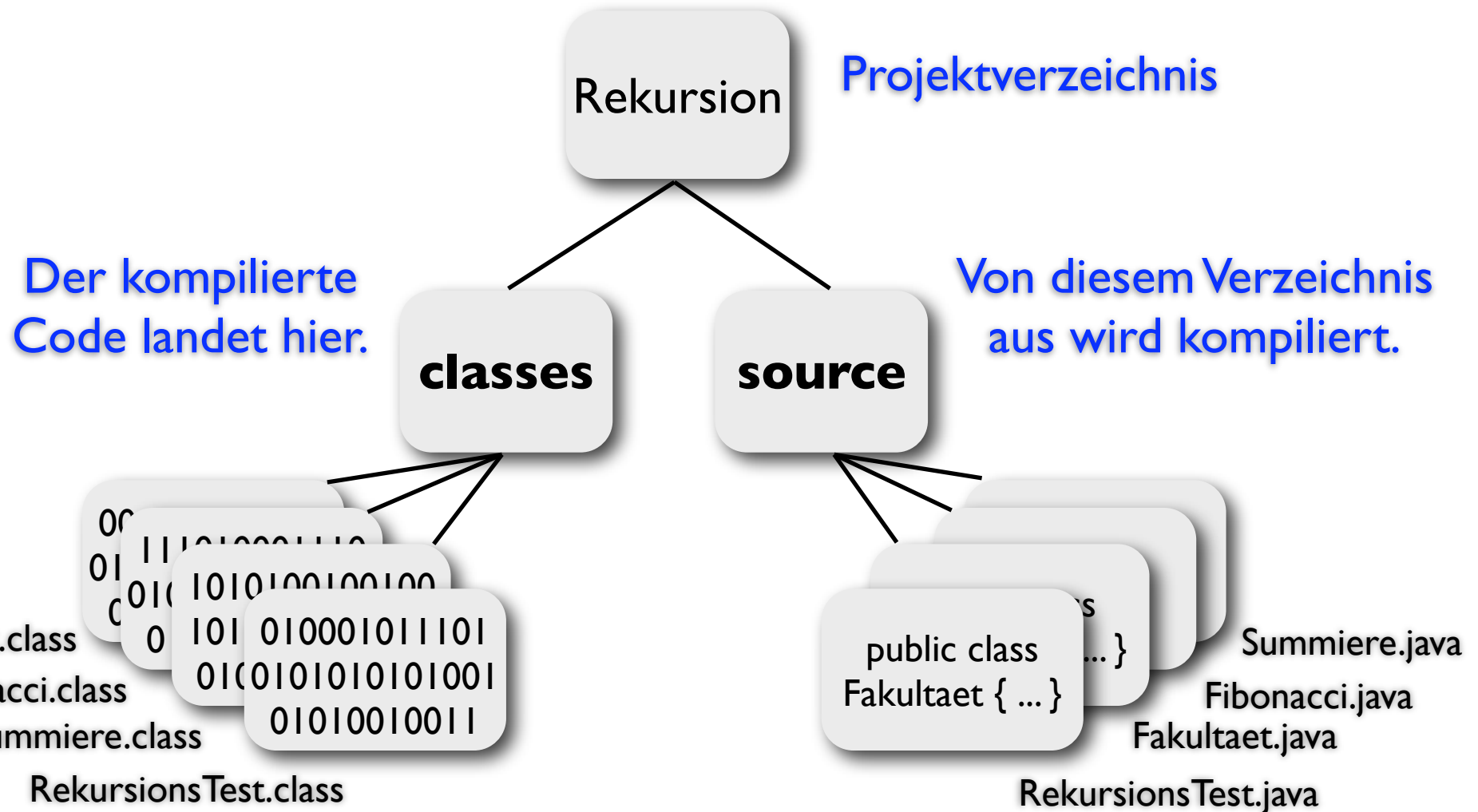
```
static public void fib (long n) {  
    if (n == 0 || n == 1) // Terminierungsfall  
        return 1;  
    else  
        return fib(n - 1) + fib(n - 2);  
}
```

(Geht übrigens auch schneller..)

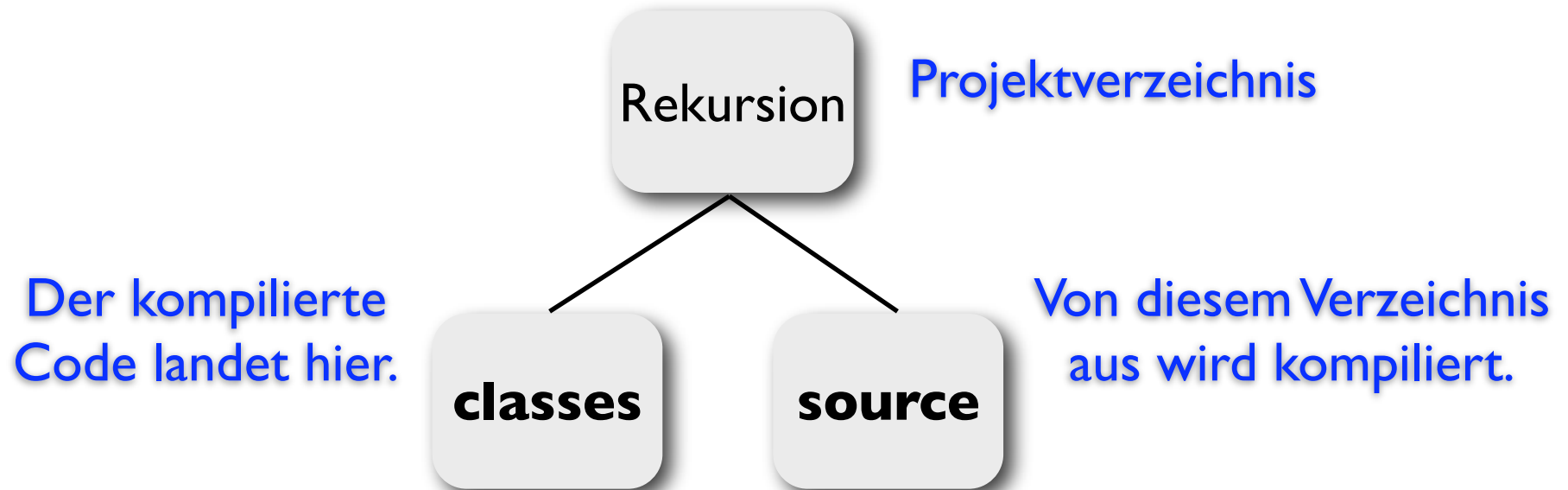
Das glorreiche Finale:
Veröffentlichen Sie Ihren Code!

Pakete und JARs

Quellcode und class-Dateien trennen

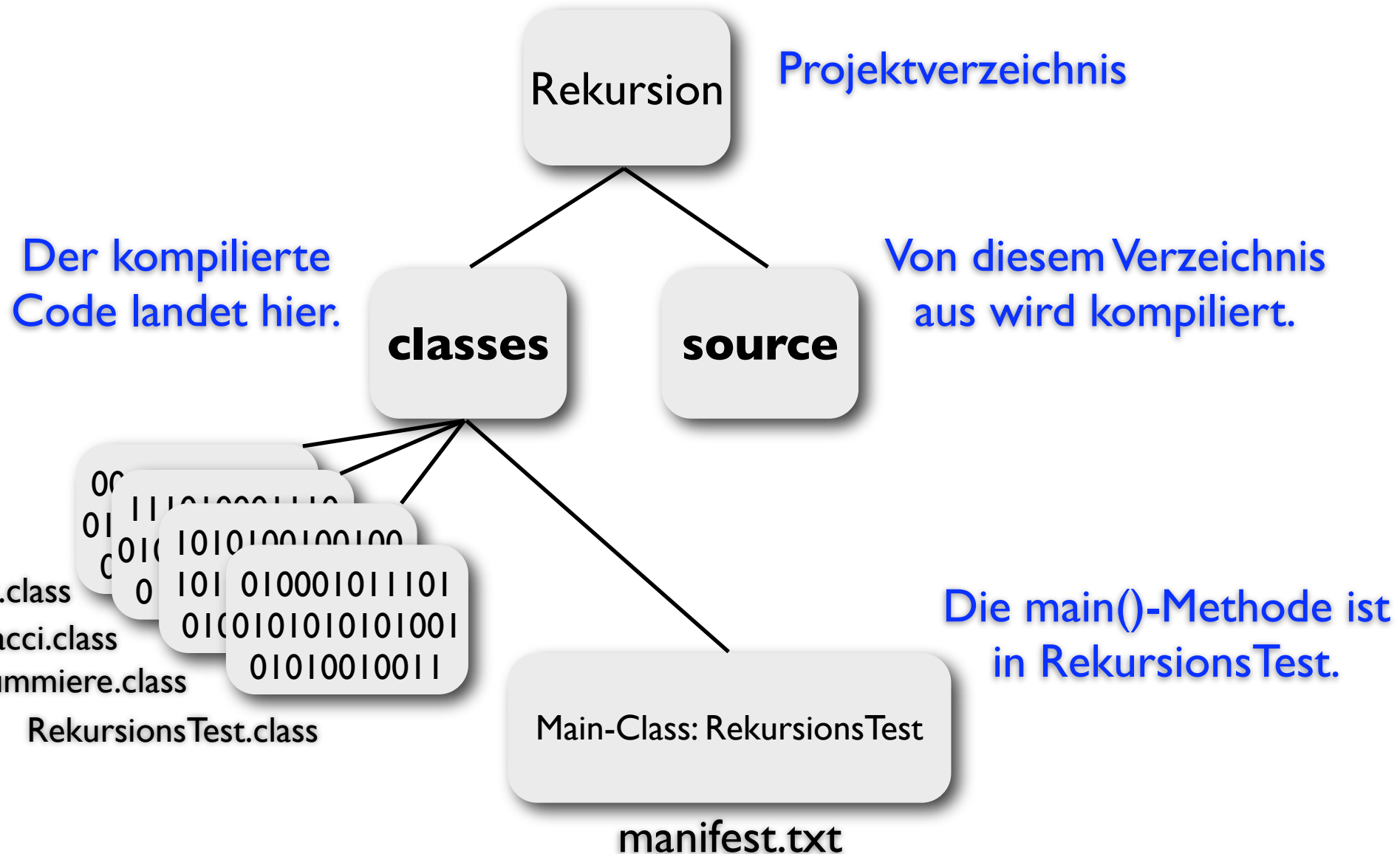


Quellcode und class-Dateien trennen



```
// Programm mit Verzeichnisschalter -d kompilieren, d.h.,  
// die class-Dateien sollen in das Verzeichnis "../classes" gespeichert werden.  
cd Rekursion/source  
javac -d ../classes RekursionsTest.java  
javac -d ../classes *.java // oder alle Dateien kompilieren  
// Programm ausführen  
cd Rekursion/classes  
java RekursionsTest
```

JAR-Dateien: Java **AR**chiv



JAR-Dateien: Java **AR**chiv



Die Manifest-Datei sagt, in welcher Klasse die main()-Methode ist.

Main-Class: RekursionsTest

manifest.txt

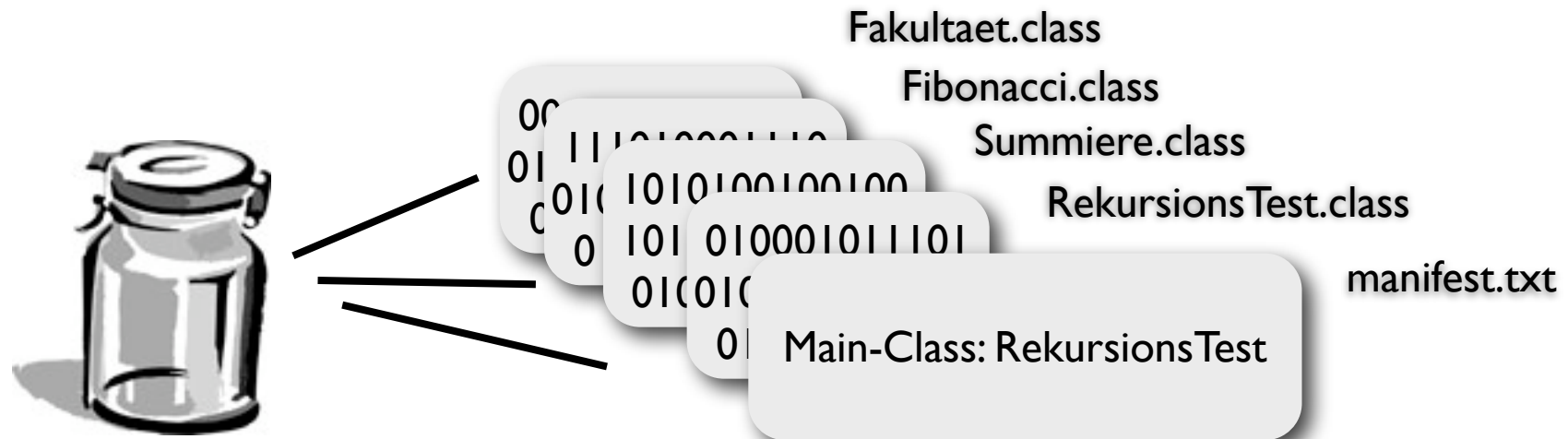
Drücken Sie *Return* nach der Main-Class-Zeile.
Schreiben Sie nicht *RekursionsTest.class*!

```
// Eine JAR-Datei erstellen, die das vollständige classes-Verzeichnis  
// UND die Datei manifest.txt enthält.
```

```
cd Rekursion/classes
```

```
jar -cvmf manifest.txt RekursionsTest.jar *.class
```

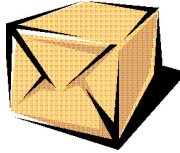
JAR-Dateien: Java **AR**chiv



// Eine JAR-Datei ausführen (evtl. auch Doppelklick möglich):

java -jar RekursionsTest.jar

Klassen und Pakete

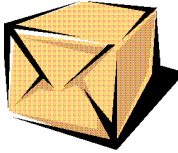


```
// vollständiger Name einer Klasse  
import java.util.ArrayList;  
import java.util.Comparator;  
import javax.swing.event.MenuKeyListener;
```



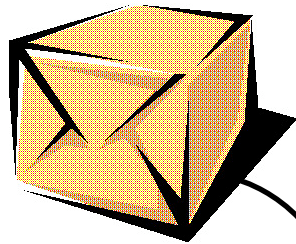
Unterschiedliche Pakete können den gleichen Namen für eine Klasse vergeben.

Klassen und Pakete



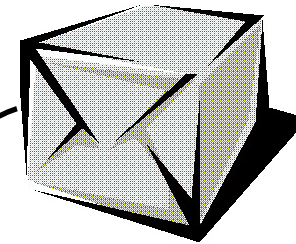
Pakete verhindern Namenskonflikte, wenn der Paketname einzigartig ist. Namenskonvention für ein Paket: **umgedrehter Domainname**

de.javavkbf



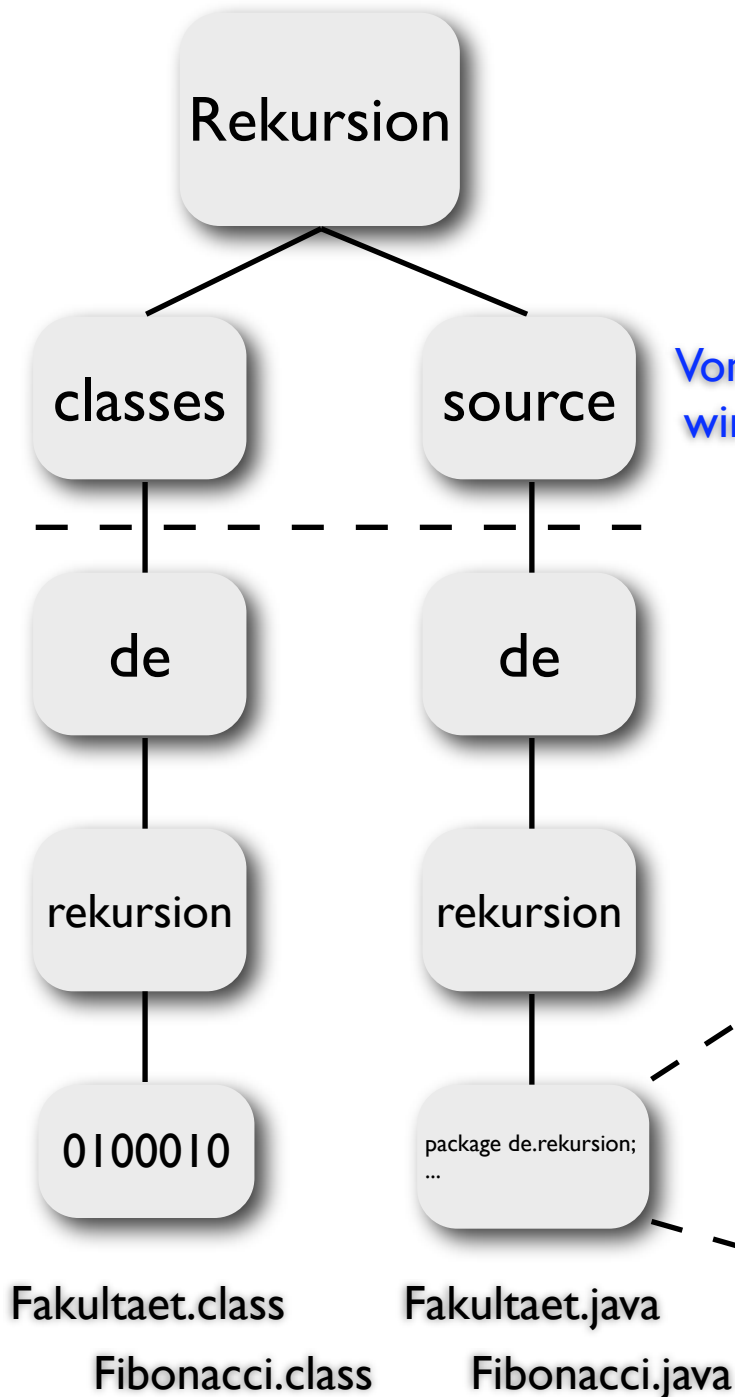
MeineKlasse

de.javafordummies

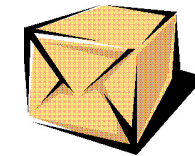


MeineKlasse

Verzeichnisstruktur für Pakete



Von diesem Verzeichnis aus wird weiterhin kompiliert.



de.rekursion

Die Verzeichnisstruktur muss der Paketstruktur entsprechen.

```
// Im Quellcode müssen wir den  
// vollständigen Paketnamen für die  
// Klasse angeben !!!
```

```
package de.rekursion;
```

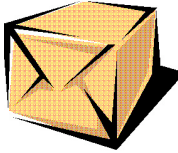
```
public class Fakultaet {
```

```
    ...
```

```
}
```

...

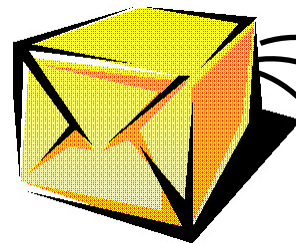
Ein Paket erstellen



```
// Der Verzeichnisschalter -d erstellt auch automatisch die
// Verzeichnisstruktur im classes-Verzeichnis
cd Rekursion/source
javac -d ../classes de/rekursion/RekursionsTest.java
javac -d ../classes de/rekursion/*.java // oder alle Dateien

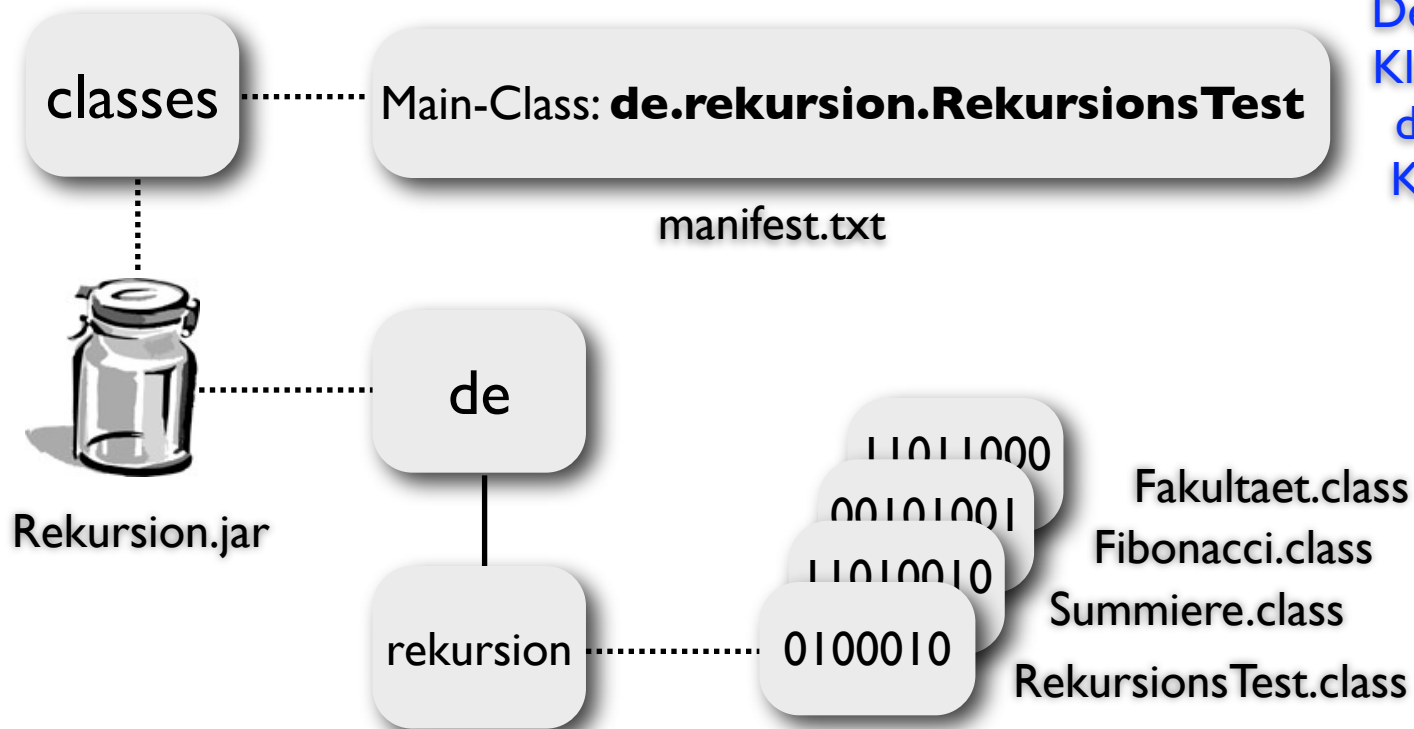
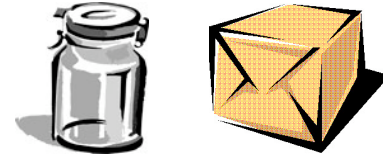
// den Code im Paket ausführen
// den vollständigen Klassennamen mit Punkt . getrennt angeben !!!
cd Rekursion/classes
java de.rekursion.RekursionsTest
```

de.rekursion



Summiere
Fibonacci
Fakultaet
RekursionsTest

Ausführbares JAR-Paket



Den vollständigen
Klassennamen für
die ausführbare
Klasse angeben!

```
// Die JAR-Datei enthält das vollständige de-Verzeichnis UND manifest.txt  
cd Rekursion/classes  
jar -cvmf manifest.txt Rekursion.jar de  
// Das JAR-Paket ausführen: RekursionsTest wird aufgerufen  
java -jar Rekursion.jar
```



Lesen Sie zu Paketen und JARs
Kapitel 17 (Seiten 581 - 595).



kann noch viel mehr !!!

Netzwerkprogrammierung

Threads

Verteilte Anwendungen

und... und... und...

Zusammenfassung

Syntax, Schleifen, Verzweigungen

Klassen, Objekte, Methoden

Variablen, Casting, Arrays, ArrayList

Parameter, Rückgabewerte, Boolesche Ausdrücke

Vererbung, Polymorphie, Interfaces

Konstruktoren, Garbage Collection

Exceptions

GUI Programmierung

Datei-E/A

Datenstrukturen

Rekursion

Pakete und JAR

Danke für's Mitmachen

und viel Spaß im weiteren Studium!