

Skript zur Vorlesung:
**Einführung in die
Programmierung**
WiSe 2009 / 2010

Skript © 2009 Christian Böhm, Peer Kröger, Arthur Zimek

Prof. Dr. Christian Böhm
Annahita Oswald
Bianca Wackersreuther

Ludwig-Maximilians-Universität München
Institut für Informatik
Lehr- und Forschungseinheit für Datenbanksysteme



14. Strukturierung von Java-Programmen: Packages

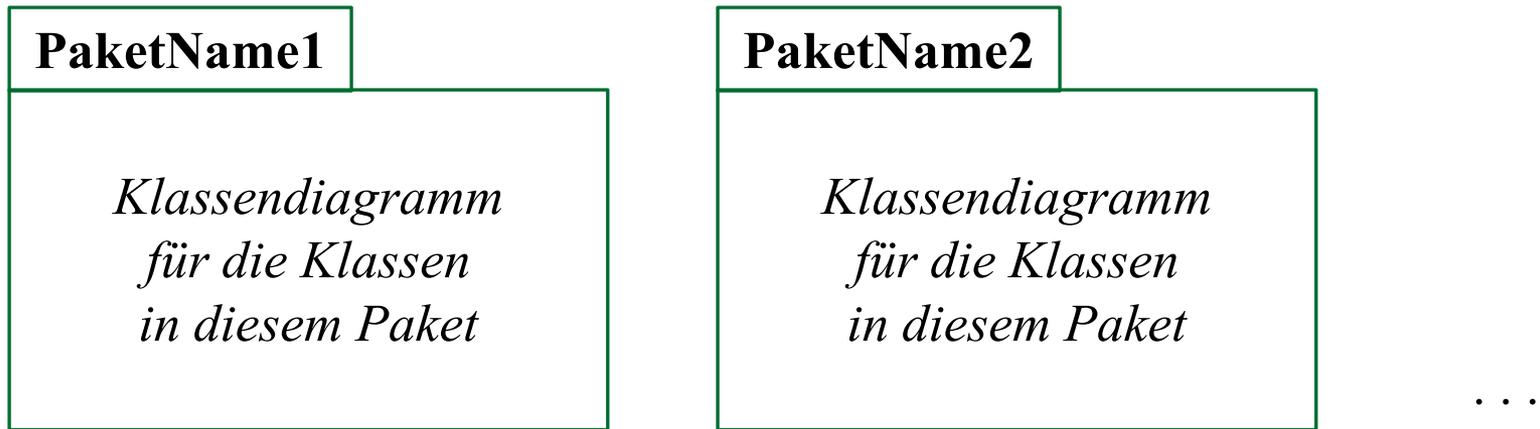
14.1 Strukturierung durch Packages

14.2 Zugriffsspezifikationen

- Bei großen Programmen entstehen viele Klassen und Schnittstellen.
- Um einen Überblick über diese Menge zu bewahren, wird ein Strukturierungskonzept benötigt, das von den Details abstrahiert und die übergeordnete Struktur verdeutlicht.
- Ein solches Strukturierungskonzept stellen die *Pakete (packages)* dar. Packages erlauben es, Komponenten zu größeren Einheiten zusammenzufassen.
- Die meisten Programmiersprachen bieten dieses Strukturierungskonzept (teilweise unter anderen Namen) an.

Packages in UML

- Packages gruppieren Klassen, die einen gemeinsamen Aufgabenbereich haben.
- In einem UML Klassendiagramm kann dies folgendermaßen notiert werden:



- Bemerkung: Es kann Beziehungen zwischen Klassen unterschiedlicher Pakete geben.
- **Welche grundlegenden OO Modellierungsaspekte werden durch das Konzept der Pakete realisiert?**

- In Java können Klassen zu Packages zusammengefasst werden.
- Packages dienen in Java dazu,
 - große Gruppen von Klassen, die zu einem gemeinsamen Aufgabenbereich gehören, zu bündeln,
 - potentielle Namenskonflikte zu vermeiden,
 - Zugriffe und Sichtbarkeit zu definieren und kontrollieren,
 - eine Hierarchie von verfügbaren Komponenten aufzustellen.

- Jede Klasse in Java ist Bestandteil von genau einem Package.
- Ist eine Klasse nicht explizit einem Package zugeordnet, dann gehört es implizit zu einem *Default*-Package.
- Packages sind hierarchisch gegliedert, können also Unterpackages enthalten, die selbst wieder Unterpackages enthalten, usw.
- Die Package-Hierarchie wird durch Punktnotation ausgedrückt:
package.unterpackage1.unterpackage2. . . .Klasse

Import von Packages

- Der vollständige Name einer Klasse besteht aus dem Klassen-Namen *und* dem Package-Namen (konventionell aus Kleinbuchstaben):
`packagename.KlassenName`
- Um eine Klasse verwenden zu können, muss angegeben werden, in welchem Package sie sich befindet.

1. Die Klasse wird entweder an der entsprechenden Stelle im Programmtext über den vollen Namen angesprochen:

```
java.util.Random einZufall = new java.util.Random();
```

2. Oder am Anfang des Programms werden die gewünschten Klassen mit Hilfe einer **import**-Anweisung eingebunden:

```
import java.util.Random;
```

```
...
```

```
Random einZufall = new Random();
```

- **Achtung:** Werden zwei Klassen gleichen Namens aus unterschiedlichen Packages importiert, müssen die Klassen trotz **import**-Anweisung mit vollem Namen aufgerufen werden!

- Klassen des Default-Packages können ohne explizite **import**-Anweisung bzw. ohne vollen Namen verwendet werden.
- Wird in der **import**-Anweisung eine Klasse angegeben, wird genau diese Klasse importiert. Alle anderen Klassen des entsprechenden Packages bleiben unsichtbar.
- Will man alle Klassen eines Packages auf einmal importieren, kann man dies mit der folgenden **import**-Anweisung:

```
import packagename.*;
```
- Achtung: Es werden dabei wirklich nur die Klassen aus dem Package `packagename` eingebunden und *nicht* etwa auch die Klassen aus Unter-Packages von `packagename`.

- Die Java-Klassenbibliothek bietet bereits eine Vielzahl von Klassen an, die alle in Packages gegliedert sind.
- Beispiele für vordefinierte Packages:

<code>java.io</code>	Ein- und Ausgabe
<code>java.util</code>	nützliche Sprach-Werkzeuge
<code>java.awt</code>	Abstract Window Toolkit
<code>java.lang</code>	Elementare Sprachunterstützung

 usw.
- Die Klassen im Package `java.lang` sind so elementar (z.B. enthält `java.lang` die Klasse `Object`, die implizite Vaterklasse aller Java-Klassen), dass sie von jeder Klasse automatisch importiert werden. Ein expliziter Import mit **`import java.lang.*;`** ist also *nicht* erforderlich.

- Ein eigenes Package `mypackage` wird angelegt, indem man vor eine Klassendeklaration und vor den `import`-Anweisungen die Anweisung `package mypackage;` platziert.
- Es können beliebig viele Klassen (jeweils aber mit unterschiedl. Namen) mit der Anweisung `package mypackage;` im selben Package gruppiert werden.
- Um Namenskollisionen bei der Verwendung von Klassenbibliotheken unterschiedlicher Hersteller zu vermeiden, ist es Konvention, für Packages die URL-Domain der Hersteller in umgekehrter Reihenfolge zu verwenden, z.B.
`com.sun.` für die Firma Sun,
`de.lmu.ifi.dbs.` für den DBS-Lehrstuhl an der LMU.

- Wie bereits erwähnt, muss die Deklaration einer Klasse x in eine Datei $x.java$ geschrieben werden.
- Darüberhinaus müssen alle Klassendeklarationen (also die entsprechenden `.java`-Dateien) eines Packages p in einem Verzeichnis p liegen.
- Beispiel:
 - Die Datei `Klasse1.java` mit der Deklaration der Klasse `package1.Klasse1` liegt im Verzeichnis `package1`.
 - Die Datei `Klasse2.java` mit der Deklaration der Klasse `package1.unterpackage1.Klasse2` liegt im Verzeichnis `package1/unterpackage1`.

- Wir hatten bereits verschiedene Schlüsselwörter zur Spezifikation der Sichtbarkeit von Klassen und Klassen-Elementen (Attributen / Methoden) kennengelernt und teilweise erweitert.
- Erst jetzt mit dem Konzept der Packages können wir allerdings alle Möglichkeiten kennenlernen.
- Im folgenden also noch einmal die (nun endgültige) Möglichkeit, die Sichtbarkeit von Klassen und Elementen einer Klasse zu spezifizieren.

Zugriffsrechte und Sichtbarkeit

- Klassen und Elemente mit der Sichtbarkeit **public** sind von allen anderen Klassen (insbesondere auch Klassen anderer Packages) sichtbar und zugreifbar.
- Klassen und Elemente mit der Sichtbarkeit **private** sind nur innerhalb der eigenen Klasse (also auch *nicht* innerhalb möglicher Unterklassen oder Klassen des selben Packages) sichtbar und zugreifbar.
- Klassen und Elemente mit der Sichtbarkeit **protected** sind innerhalb des gesamten Packages und aller Unterklassen (auch außerhalb des Packages) sichtbar und zugreifbar.
- Klassen und Elemente, deren Sichtbarkeit *nicht* durch ein entsprechendes Schlüsselwort spezifiziert ist, erhalten per Default die sogenannte *package scoped (friendly)* Sichtbarkeit: Diese Elemente sind nur für Klassen innerhalb des selben Packages sichtbar und zugreifbar.

Spezifikation	sichtbar in			
	allen Klassen	allen Unterklassen unabh. vom Package	Klassen im selben Package	selber Klasse
public	✓	✓	✓	✓
protected		✓	✓	✓
			✓	✓
private				✓