

12. Strukturierung von Java-Programmen: Packages

12.1 Strukturierung durch Packages

12.2 Zugriffsspezifikationen

12.3 Zusammenfassung

12. Strukturierung von Java-Programmen: Packages

12.1 Strukturierung durch Packages

12.2 Zugriffsspezifikationen

12.3 Zusammenfassung

- 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.

- In Java können Klassen und Interfaces zu Packages zusammengefasst werden.
- Packages dienen in Java dazu,
 - große Gruppen von Klassen und Interfaces, 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 und jedes Interface in Java ist Bestandteil von genau einem Package.
- Ist eine Klasse oder ein Interface nicht explizit einem Packet 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`

- Der vollständige Name einer Klasse (analog: eines Interfaces) besteht aus dem Klassen-Namen *und* dem Package-Namen:
`packagename.KlassenName`
- Package-Namen bestehen nach Konvention immer aus Kleinbuchstaben.
- Um eine Klasse oder ein Interface verwenden zu können, muss angegeben werden, in welchem Package es sich befindet. Dies kann auf zwei Arten geschehen:
 - ① Die Klasse/das Interface wird an der entsprechenden Stelle im Programmtext über den vollen Namen angesprochen:
`java.util.Random einzufall = new java.util.Random();`
 - ② 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:
 `java.io` Ein- und Ausgabe
 `java.util` nützliche Sprach-Werkzeuge
 `java.awt` Abstract Window Toolkit
 `java.lang` Elementare Sprachunterstützung
 usw.
- Die Klassen und Interfaces 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 Klassen- bzw. Interfacedeklaration und vor den `import`-Anweisungen die Anweisung `package mypackage;` platziert.
- Es können beliebig viele Klassen und Interfaces (jeweils aber mit unterschiedlichen 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 bzw. eines Interfaces `x` in eine Datei `x.java` geschrieben werden.
- Darüberhinaus müssen alle Klassen und Interface-Deklarationen (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`.

12. Strukturierung von Java-Programmen: Packages

12.1 Strukturierung durch Packages

12.2 Zugriffsspezifikationen

12.3 Zusammenfassung

- 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.
- Zur Erinnerung: Die Methoden und Konstanten von Interfaces sind grundsätzlich **public**.

- 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				✓

12. Strukturierung von Java-Programmen: Packages

12.1 Strukturierung durch Packages

12.2 Zugriffsspezifikationen

12.3 Zusammenfassung

Sie kennen jetzt

- das Konzept der Packages,
- den vollständigen Namen von Klassen und Interfaces,
- Möglichkeiten, um Klassen und Interfaces aus anderen Paketen zu verwenden,
- Möglichkeiten, um Klassen und Interfaces in eigenen Packages zusammenzufassen,
- Möglichkeiten, die Sichtbarkeit von Klassen und Elementen einer Klasse zu spezifizieren.