Abschnitt 13: Strukturierung von Java-Programmen: Packages

13. Strukturierung von Java-Programmen: Packages
13.1 Strukturierung durch Packages
13.2 Zugriffsspezifikationen

Überblick

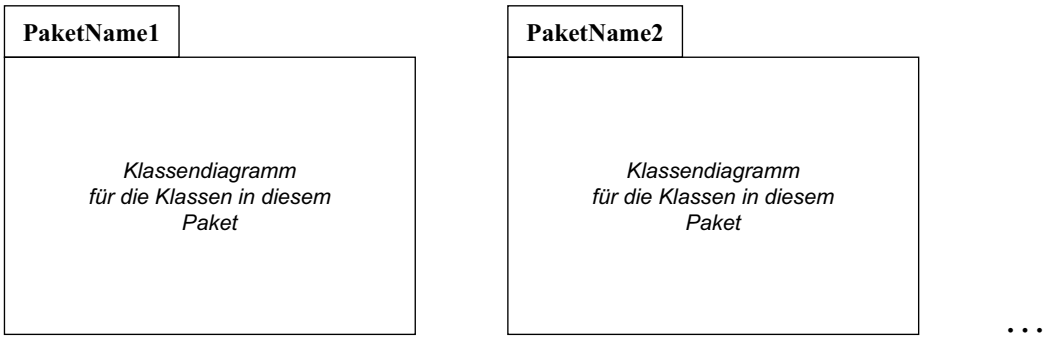
13. Strukturierung von Java-Programmen: Packages
13.1 Strukturierung durch Packages
13.2 Zugriffsspezifikationen

Pakete in der Software-Entwicklung

- 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?

Packages in Java

- 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.

Packages in Java

- 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: `packagename.KlassenName`
- Package-Namen bestehen nach Konvention immer aus Kleinbuchstaben.
- Um eine Klasse verwenden zu können, muss angegeben werden, in welchem Package sie sich befindet. Dies kann auf zwei Arten geschehen:
  1. Die Klasse 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!

Import von Packages

- 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`.

Packages aus der Java-Klassenbibliothek

- 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 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.

Deklaration eigener Packages

- 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 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.

Deklaration eigener Packages

- 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`.

Überblick

13. Strukturierung von Java-Programmen: Packages
13.1 Strukturierung durch Packages
13.2 Zugriffsspezifikationen

Zugriffsrechte und Sichtbarkeit

- 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 nocheinmal 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.

Zugriffsrechte und Sichtbarkeit

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