9 Pakete und Module

9.1 Grundsätzliches

- *Pakete* (*packages*) dienen zur Gruppierung logisch zusammengehörender Typen (Klassen und Schnittstellen). Logisch zusammengehörende Pakete können wiederum in *Modulen* gruppiert werden, die eventuell nur einen Teil ihrer Pakete *exportieren*.
- Jedes Paket stellt einen eigenen *Namensbereich* dar, d. h. unterschiedliche Pakete können unterschiedliche Typen (und Teilpakete; vgl. § 9.2) mit dem gleichen (unqualifizierten) Namen enthalten.
  Die Namen aller Typen und Teilpakete eines Pakets müssen jedoch eindeutig sein.
- Typen können öffentlich (Schlüsselwort `public`) oder paketöffentlich (ohne Angabe einer Zugriffsbeschränkung) sein (vgl. § 4.3).
  Paketöffentliche Typen können nur innerhalb desselben Pakets verwendet werden, öffentliche Typen in allen Paketen.
- Elemente eines Typs, die paket- oder unterklassenöffentlich sind, können in allen Typen desselben Pakets verwendet werden.
- Um Typen aus anderen Paketen anzusprechen, muss ihr Name entweder mit dem vollständigen Namen des Pakets *qualifiziert* werden (z. B. `java.io.InputStream`), oder der Name muss zuvor *importiert* werden (vgl. § 9.3).

9.2 Paketnamen

- Paketnamen bestehen aus einem oder mehreren *einfachen Namen*, die durch Punkte getrennt sind, z. B. `java`, `java.awt`, `java.awt.color`.
- Ein Paket mit dem Namen `P.Q` (wobei `P` ein vollständiger Paketname und `Q` ein einfacher Name ist) ist ein *Teilpaket* (*subpackage*) des Pakets `P`.
- Für ein Teilpaket gelten dieselben Zugriffsrechte wie für andere Pakete, d. h. es darf ebenfalls nur die öffentlichen Typen seiner über- und untergeordneten Pakete verwenden.
- Paketnamen (wie z. B. `java.awt.color`) werden normalerweise auf entsprechende Verzeichnisnamen (im Beispiel `java/awt/color`) abgebildet.
- Es gibt bestimmte Namenskonventionen für Paketnamen, bei deren Einhaltung weltweit sichergestellt ist, dass Code von unterschiedlichen Entwicklern keine gleichnamigen Pakete enthält.
- Insbesondere sollten die Namen öffentlich zur Verfügung gestellter Pakete immer mit dem (umgedrehten und ggf. geeignet angepassten) Internet-Domainnamen der erstellenden Organisation beginnen, z. B. `de.hs_aalen.informatik`.
- Innerhalb einer Organisation sollte es zusätzliche Regeln zur Bildung eindeutiger Paketnamen geben.

9.3 Import-Deklarationen

- Um Typen aus anderen Paketen mit ihren einfachen Namen anzusprechen zu können, müssen sie zuvor mit Hilfe von `import`-Deklarationen bekanntgemacht werden.

- Eine Deklaration der Art `import P.T` (mit einem vollständigen Paketnamen `P` und einem einfachen Namen `T`, z. B. `import java.io.InputStream`) importiert den öffentlichen Typ `T` aus dem Paket `P` („Einzelimport“), d. h. anschließend kann der Typ `P.T` mit seinem einfachen Namen `T` angesprochen werden.

- Eine Deklaration der Art `import P.*` (mit einem vollständigen Paketnamen `P`, z. B. `import java.io.*`) importiert alle öffentlichen Typen aus dem Paket `P` („Massenimport“), d. h. anschließend können alle diese Typen mit ihren einfachen Namen angesprochen werden.

- Eine Deklaration der Art `import P.T.N` bzw. `import P.T.*` (mit einem vollständigen Paketnamen `P` und einfachen Namen `T` und `N`) importiert den zugreifbaren geschachtelten Typ `N` bzw. alle zugreifbaren geschachtelten Typen, die im zugreifbaren Typ `P.T` definiert sind.

- Die Typen des vordefinierten Pakets `java.lang` müssen nicht explizit importiert werden (d. h. jede Quelldatei enthält automatisch eine Deklaration `import java.lang.*`).

- Namen aus Massenimporten können von einzeln importierten Namen sowie von Namen innerhalb des aktuellen Pakets verdeckt werden, zum Beispiel:

```
package test;
import java.sql.Date;
import java.util.*;
class List { ..... }
```

  Hier bezeichnet `Date` den Typ `java.sql.Date` (und nicht `java.util.Date`) und `List` den Typ `test.List` (und nicht `java.util.List`).

- Massenimporte sind zwar bequem zu schreiben, helfen dem Leser aber nicht bei der Suche nach importierten Namen, zum Beispiel:

```
import java.awt.*;
import javax.swing.*;
```

  Aus welchem Paket stammen die Namen `Box` und `Button`?

9.4 Statische Import-Deklarationen

- Eine Deklaration der Art `import static P.T.x` (mit einem vollständigen Paketnamen `P` und einfachen Namen `T` und `x`, z. B. `import static java.lang.Math.PI`) importiert alle zugreifbaren statischen Elemente (Felder und Methoden sowie geschachtelte Klassen und Schnittstellen) mit dem Namen `x`, die im zugreifbaren Typ `P.T` definiert sind, d. h. anschließend können diese mit ihrem einfachen Namen `x` angesprochen werden.

- Eine Deklaration der Art `import static P.T.*` (mit einem vollständigen Paketnamen `P` und einem einfachen Namen `T`, z. B. `import static java.lang.Math.*`) importiert alle zugreifbaren statischen Elemente, die im zugreifbaren Typ `P.T` definiert sind, d. h. anschließend können diese jeweils mit ihrem einfachen Namen angesprochen werden.

- Wenn nur der geschachtelte Typ `x` bzw. alle geschachtelten Typen, die im Typ `P.T` definiert sind, importiert werden sollen, kann die Deklaration `import P.T.x` bzw. `import P.T.*` verwendet werden (vgl. § 9.3).

9.5 Quelldateien

- Am Anfang einer Quelldatei (compilation unit) kann der vollständige Name des Pakets, zu dem die in der Datei definierten Typen gehören sollen, mit Hilfe einer `package`-Deklaration vereinbart werden, z. B.:

  ```
  package de.hs_aalen.informatik.heinlein.test;
  ```

  Wenn kein Paket vereinbart wird, gehören die Typen zu einem anonymen Paket.

- Anschließend können beliebig viele `import`-Deklarationen in beliebiger Reihenfolge angegeben werden, z. B.:

  ```
  import java.util.List;          // Einzelimport.
  import java.awt.*;              // Massenimport.
  ```

- Anschließend können beliebig viele Typen definiert werden, von denen normalerweise maximal einer öffentlich sein darf.
  Wenn ein öffentlicher Typ definiert wird, muss der Name der Quelldatei normalerweise mit dem Namen des Typs (plus Erweiterung `.java`) übereinstimmen.

- Alle o. g. Bestandteile einer Quelldatei können auch fehlen, d. h. prinzipiell könnte eine Quelldatei auch leer sein.

- Typdefinitionen können sich beliebig gegenseitig (auch zirkulär) referenzieren, auch über Paket- und Quelldateigrenzen hinweg.