Module

- Definition von Modulen
- Importieren von Modulen
- Wichtige Standard-Module von Java

- Packages, Imports, CLASSPATH

Module

In Java können wir mehrere Übersetzungseinheiten zu einem **Modul** oder **Paket** (engl. *package*) zusammenfassen. Ein Modul ist eine Sammlung von Klassen, die inhaltlich zusammengehören und die von anderen Klassen benutzt werden können.

Definition von Modulen

Die Übersetzungseinheiten, die man in einem gemeinsamen Modul zusammenfassen will, müssen alle in demselben Dateiverzeichnis abgespeichert werden.

Um eine Übersetzungseinheit zu einem Mitglied eines Moduls zu machen, muss am Dateianfang eine **Package-Deklaration** stehen, die folgende Syntax hat:

package ***Packagename*** **;**

Ein **Packagename** ist ein **Name**.

Ein **Name** ist entweder ein **Bezeichner** oder ein **qualifizierter Name**.

Ein **qualifizierter Name** ist eine durch Punkte (**.**) miteinander verbundene Folge von Bezeichnern.

Module

Definition von Modulen

Beispiel: Package `MathPack`

```
package MathPack;

public class Fakultaet {
   static public long fakultaet(long n) {
      if (n <= 0)
         return 1;
      else
         return n * fakultaet(n-1);
   }
}
```

Die Datei `Fakultaet.java` muss in einem Dateiverzeichnis namens `MathPack` stehen.

Weitere Übersetzungseinheiten in diesem Dateiverzeichnis, die ebenfalls mit der Anweisung

```
package MathPack;
```

beginnen, gehören auch zum Modul `MathPack`.

Module

Importieren von Modulen

Pakete *exportieren* die öffentlichen (`public`) Klassen, die vom Benutzer dann *importiert* werden können.

Auf eine öffentliche Methode einer öffentlichen Klasse eines Moduls z. B. können wir immer über den vollständig qualifizierten Namen der Methode zugreifen.

Beispiel: Zugriff über vollständig qualifizierten Namen

```
public class FakultaetTest {
  public static void main(String[] argv){
    long n = 6;
    System.out.println(n + "! = " +
      MathPack.Fakultaet.fakultaet(n));
  }
}
```

Module

Importieren von Modulen

Häufig ist eine verkürzte Schreibweise für den Zugriff auf einen Klassennamen wünschenswert. Hierzu dient die **Import-Deklaration**.

Es gibt zwei Formen der Import-Deklaration:

```
import Packagename.Klassenname ;
import Packagename.* ;
```

Die erste Form macht die angegebene Klasse des angegebenen Pakets allein über den Klassennamen verfügbar.

Beispiel: Import per Klassenname

```
import MathPack.Fakultaet;

public class FakultaetTest1 {
  public static void main(String[] argv){
    long n = 6;
    System.out.println(
      n + "! = " + Fakultaet.fakultaet(n));
  }
}
```

Module

Importieren von Modulen

Die zweite Form macht *alle* Klassen des Pakets über den Klassennamen verfügbar.

Beispiel: Import aller Klassen eines Moduls

```
import MathPack.*;

public class FakultaetTest2
{
  public static void main(String[] argv) {
    long n = 6;
    System.out.println(
      n + "! = " + Fakultaet.fakultaet(n));

    Elch drElch = new Elch("DOS");
    drElch.moeh();
  }
}
```

Module

Importieren von Modulen

Wie findet der Java-Compiler die Klasse `Fakultaet`?

Alle Übersetzungseinheiten, die zum Modul `MathPack` gehören, müssen in dem gleichnamigen Dateiverzeichnis abgespeichert werden.

Der Punkt der `import`-Anweisung trennt also den Verzeichnisnamen vom Klassennamen.

Der Java-Compiler sucht im Verzeichnis `MathPack` nach der Klasse `Fakultaet`. Aber wo im Verzeichnisbaum beginnt er nach diesem Verzeichnis zu suchen?

Dies wird in der Umgebungsvariablen **`CLASSPATH`** festgelegt.

Module

Importieren von Modulen

Ist CLASSPATH z. B. folgendermaßen definiert:

```
export CLASSPATH=/vorlesung/bsp/Java05:.:..
```

dann wird das Unterverzeichnis `MathPack` in den Verzeichnissen

- `/vorlesung/bsp/Java05`
- dem aktuellen Arbeitsverzeichnis (`.`)
- dem "Vaterverzeichnis" des aktuellen Arbeitsverzeichnisses (`..`)

gesucht, und zwar in dieser Reihenfolge.

Die folgenden Kommandos erweitern jeweils die Variable CLASSPATH um das aktuelle Arbeitsverzeichnis.

unter DOS/Windows:

```
set CLASSPATH=%CLASSPATH%;.
```

unter UNIX (Korn-Shell):

```
export CLASSPATH=$CLASSPATH:.
```

Module

Importieren von Modulen

Implizit ist in allen Java-Quelldateien die folgende Import-Deklaration enthalten:

```
import java.lang.*;
```

Daher können wir verkürzt

`System.out.println`

für die Methode

`java.lang.System.out.println`

schreiben.

Warum weist der Compiler die folgende Import-Deklaration als fehlerhaft ab?

```
import System.out;
```

Module

Importieren von Modulen

Ein Verzeichnis kann neben Übersetzungseinheiten, die Klassen für ein Modul beschreiben, auch Unterverzeichnisse enthalten, die ihrerseits wieder Module enthalten. Damit können wir eine Hierarchie von Modulen aufbauen.

Beispiel:

Angenommen der Pfad zur Datei `d.java` ist
"`/a/b/c/d.java`".

Die letzte Komponente ergibt den Namen der öffentlichen Klasse, nämlich `d`.
(z. B.: `public class d {... )`

Der Name des Moduls wird aus dem Rest des Pfadnamens gebildet, wobei die Trennzeichen `/` (bzw. `\` unter DOS/Windows) durch Punkte ersetzt werden.

Wenn die Übersetzungseinheit mit einer `package`-Anweisung anfängt, dann gibt es in diesem Beispiel nur drei Möglichkeiten:

```
package a.b.c;
package b.c;
package c;
```

Module

Importieren von Modulen

Beziehung von CLASSPATH, Modulname und Klassenname zum Dateisystem

Beispiel:

Die drei Zeilen der folgenden Tabelle

CLASSPATH	Modulname	Klassenname
`/alpha/beta/gamma`	`delta`	`epsilon`
`/alpha/beta`	`gamma.delta`	`epsilon`
`/alpha`	`beta.gamma.delta`	`epsilon`

bezeichnen alle die Datei

```
/alpha/beta/gamma/delta/epsilon.java
```

Module

Importieren von Modulen

Kommt in zwei verschiedenen Modulen `Pack1` und `Pack2` jeweils eine Klasse mit einem bestimmten Namen vor, etwa `myClass`, so kann man auf beide zugreifen, da jeweils davor der Modulname geschrieben werden muss und diese Klassen damit "von außen" unterschiedliche Bezeichner tragen.

Module legen also jeweils einen eigenen *Namensraum* fest.

Die Java-Entwickler haben ein Internet-weit gültiges eindeutiges Schema für die Benennung von Modulen vorgeschlagen. Dieses Schema beruht auf dem Internet-Domain-Namen der Organisation, bei der das Paket entwickelt wurde.

Beispiele:

```
DE.fhtrier.informatik.kuenkler.javatools.
utilities.TastaturEingabe.readInt()

com.sun.fruits.peach.eat()

java.lang.String.charAt()
```

Module

Wichtige Standard-Module von Java

- **`java.applet`**:
  - Unterstützung für Applet-Programmierung, Sound-Unterstützung

- **`java.awt`**:
  - (abstract windowing toolkit) grafische Benutzeroberflächen

- **`java.io`**:
  - Ein-/Ausgabe

- **`java.lang`**:
  - die wichtigsten Basisklassen wie Zahlen, Zeichenketten, Objekte, Systemfunktionen, Threads

- **`java.net`**:
  - Klassen, die für Netzwerkanwendungen wichtig sind; Zugriff auf URLs

- **`java.util`**:
  - weitere nützliche Klassen wie `Date`, `Time`, Keller, Bitmengen, Hash-Tabellen, Wörterbücher

Wichtige Standard-Module von Java

Beispiel: Verwendung der Klasse `Date`

```
import java.util.Date;

class Date2 {
  public static void main(String[] args) {
    Date now = new Date();
    System.out.println(now);
  }
}
```

Ausgabe des Programms:

```
Mon Jun 07 21:31:49 GMT+00:00 1999
```

Packages, Imports, CLASSPATH

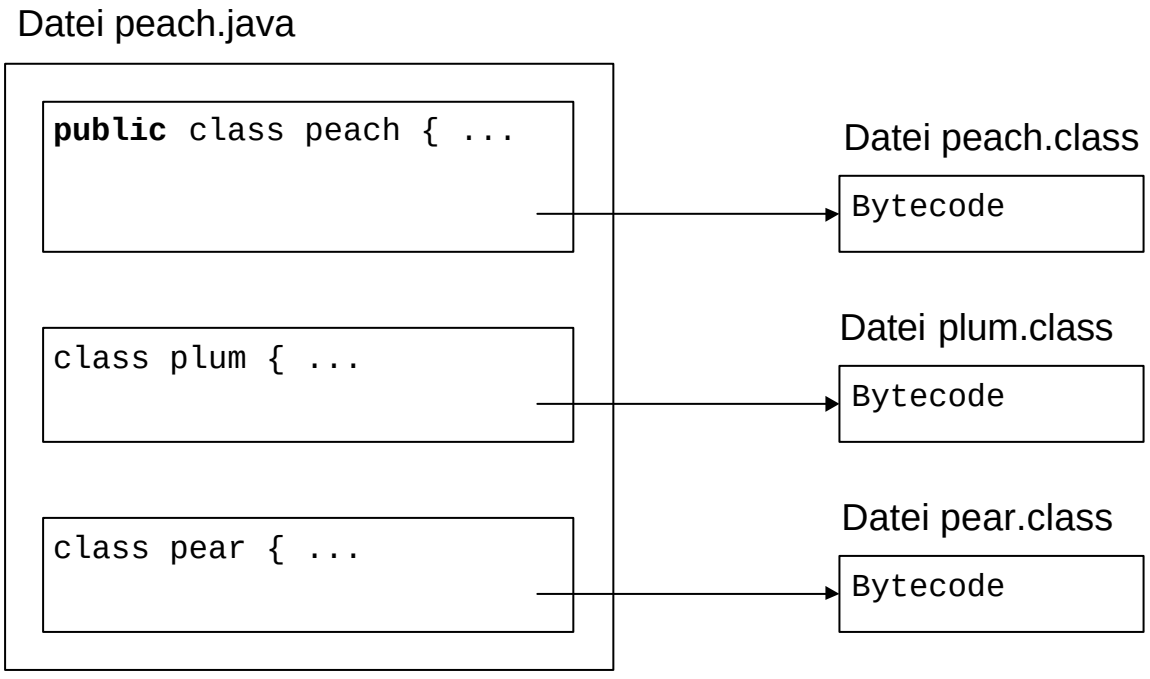
Der Dateiname bezieht sich auf den Klassennamen.

Eine Java-Quelldatei besteht (unter anderem) aus Klassendeklarationen.

Für jede Klasse einer `.java`-Datei erzeugt der Java-Compiler eine eigene `.class`-Datei, die den entsprechenden Bytecode enthält.

Höchstens eine Klasse in der Datei kann als "öffentlich" (engl. *public*) deklariert werden. Der Name dieser Klasse muss dem Dateinamen entsprechen.

Z. B. muss "`public class peach`" in der Datei `peach.java` stehen.

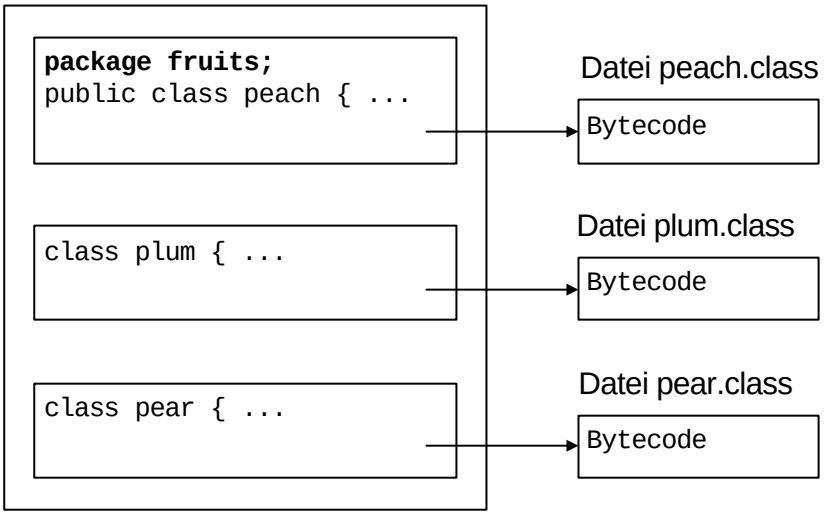


Module

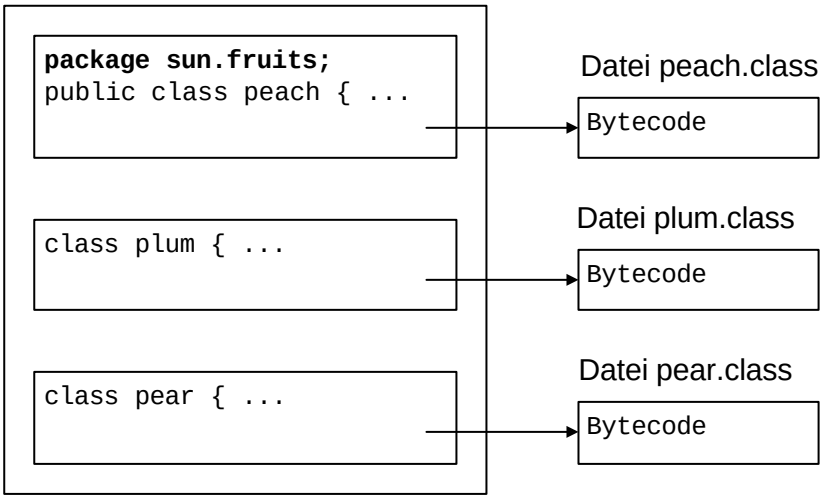
Packages, Imports, CLASSPATH

Wie die Namen der Module mit den Verzeichnisnamen zusammenhängen.

/sun **/fruits** /peach.java



/sun **/fruits** /peach.java



Module

Packages, Imports, CLASSPATH

Der Zusammenhang zwischen Importnamen, Modulnamen und Verzeichnisnamen.