CARL VON OSSIETZKY universität OLDENBURG

Teil

Objektorientierte Programmierung

Unterrichtseinheit 28

Pakete

Dr. Dietrich Boles

Gliederung

- Motivation
- Definition von Paketen
- Nutzung von Paketen
- Statischer Import
- CLASSPATH
- jar
- Umgang mit Paketen
- Pakete in Eclipse
- Zusammenfassung

Motivation

- Im Allgemeinen bestehen Programme aus vielen, vielen Klassen
- bestimmte Programmteile (Klassen) werden häufig gebraucht ($\rightarrow$ Speicher)
  - im Allgemeinen sogar in verschiedenen Anwendungen
  - im Allgemeinen sogar von unterschiedlichen Programmierern
- **gesucht**:
  - Hilfsmittel, das es einem Programmierer erlaubt, seine Klassen übersichtlich und strukturiert abspeichern und verwalten zu können
  - Hilfsmittel, das es einem Programmierer erlaubt, von ihm erstellte Klassen für mehrere Anwendungen nutzen zu können bzw. auch anderen Programmierern zur Verfügung stellen zu können
- **Java**: $\rightarrow$ Pakete (Packages)
- **Klassenbibliothek**: Sammlung von nützlichen, häufig gebrauchten Klassen, die (anderen) Programmierern zur Verfügung gestellt werden
- **Java-Packages**: Hilfsmittel zur Strukturierung von Klassenbibliotheken

Definition von Paketen (1)

- Schlüsselwort: **`package`**
- package-Anweisung: **`package <paketname>;`**

- Beispiel:
  - Datei: SchachBrett.java
  - Datei: SchachFigur.java
  - Datei: SchachSpieler.java
  - Datei: SchachRegeln.java
  - Datei: SchachSpielzug.java

- → Paket: schach
- in jeder der obigen Dateien muss als erste Anweisung (!!!!) die folgende package-Anweisung stehen: **`package schach;`**
- Voraussetzung: Bereitgestellte Klassen und Methoden müssen als **`public`** deklariert sein (siehe auch UE 30)

Definition von Paketen (2)

Datei: SchachBrett.java

```
package schach;
public class SchachBrett {
...
}
```

Datei: SchachSpieler.java

```
package schach;
public class SchachSpieler{
...
}
```

Datei: SchachFigur.java

```
package schach;
public class SchachFigur {
...
}
```

Datei: SchachRegeln.java

```
package schach;
public class SchachRegeln {
...
}
```

Definition von Paketen (3)

- Anmerkungen:
  - Der Paketname ist ein Java-Bezeichner
  - Die Dateien/Klassen eines Paketes müssen sich alle in demselben Verzeichnis befinden!
  - In einem Verzeichnis kann nur ein einziges Paket definiert werden!
  - Der Name eines Verzeichnisses, in dem ein Paket definiert wird, muss gleich dem Namen des Paketes sein!
  - Wird in einem Paket **X** eine Klasse **Y** definiert, so lautet der **vollständige Name** der Klasse **X.Y**
  - Pakete lassen sich strukturieren (Punkt-Notation verwenden!)

Strukturierung:	**Verzeichnisstruktur:**	**Pakete**
- Spiele	**spiele**	**package spiele;**
- Reversi	**spiele/reversi**	**package spiele.reversi;**
- dame	**spiele/dame**	**package spiele.dame;**
- Schach	**spiele/schach**	**package spiele.schach;**

Nutzung von Paketen (1)

- Schlüsselwort: **`import`**
- import-Anweisung: **`import <paket-qualifier>;`**
- Mehrere import-Anweisungen möglich
- Import-Anweisungen müssen hinter einer evtl. vorhandenen package-Anweisung, aber vor dem Rest des Programms stehen

- Es bestehen vier verschiedene Möglichkeiten, Dateien/Klassen zu importieren bzw. auf die Elemente der Dateien/Klassen zuzugreifen:
  - kein expliziter Import
  - Import einzelner Dateien/Klassen des Paketes
  - Import des Paketes
  - Import aller Dateien/Klassen des Paketes

- Beispiel:
  - Paket: **`java.util`**
    - Datei/Klasse: **`Date`**
    - Datei/Klasse: **`Stack`**
    - Datei/Klasse: **`Vector`**

Nutzung von Paketen (2)

- kein expliziter Import (Zugriff über **vollständigen Namen**)

```
// kein import
...
java.util.Date date = new java.util.Date();
java.util.Vector vector = new java.util.Vector();
```

- Import einzelner Dateien/Klassen des Paketes (**empfohlen**!):

```
import java.util.Date;
...
Date date = new Date();
Vector vector = new Vector(); // Fehler!
```

Nutzung von Paketen (3)

- Import aller Dateien/Klassen des Paketes (nicht der Unterpakete!):

```
import java.util.*;
...
Date date = new Date();
Vector vector = new Vector();
```

- Import des Paketes (unüblich):

```
import java.util;
...
util.Date date = new util.Date();
util.Vector vector = new util.Vector();
```

Nutzung von Paketen (4)

- Namenskonflikte (Beispiel):
  - **`package util;`** → **`class Vector`**
  - **`package misc;`** → **`class Vector`**

- eigenes Programm:

```
import util.*;
import misc.*;
...
Vector v = new Vector(); //Fehler: welcher Vector?

// korrekt (Zugriff über vollständigen Namen):
util.Vector v1 = new util.Vector();
misc.Vector v2 = new misc.Vector();
```

Nutzung von Paketen (5)

- JDK-Paket: **`java.lang`**
  - Datei/Klasse: **`System`**
  - Datei/Klasse: **`Object`**
  - Datei/Klasse: **`String`**
  - ...
  - $\rightarrow$ import-Anweisung ist **nicht** notwendig (implizites `import`)

- Anonyme Pakete:
  - Fehlt in Dateien eines Verzeichnisses die `package`-Anweisung, dann bilden die Dateien ein so genanntes "**anonymes Paket**"
  - der Zugriff auf die Elemente eines anonymen Paketes ist ausschließlich auf Dateien im selben Verzeichnis (also Dateien/Klassen des anonymen Paketes selbst) beschränkt!

Statischer Import

- Zugriff auf statische Elemente einer Klasse:

```
<Klassenname>.<Elementname>
```

- Statischer Import (ab Java 5.0):

```
import static <vollständiger Elementname>;
```

- Zugriff: **<Elementname>**

- Beispiel:

```
import static java.lang.Math.sin;
import static java.lang.Math.PI;
...
System.out.println(sin(PI));
//System.out.println(Math.sin(Math.PI));
```

CLASSPATH (1)

- CLASSPATH: Variable der Betriebssystem-Shell

- UNIX:
  - Setzen:
    - csh: **`setenv CLASSPATH ".:/user/fb10/dibo/java"`**
    - bash: **`export CLASSPATH=.:/user/fb10/dibo/java`**
  - Abfrage: **`echo $CLASSPATH`**

- Windows:
  - Setzen:

    `set CLASSPATH=.;e:\java`
  - Abfrage:

    `echo %CLASSPATH%`

CLASSPATH (2)

wichtig:

- In den CLASSPATH müssen die Verzeichnisse (bzw. zip- oder jar-Dateien) aufgenommen werden, in denen der Java-Compiler und -Interpreter nach Paketen suchen soll
- Trennung mehrerer Verzeichnisse durch einen Doppelpunkt (:) unter UNIX bzw. durch Semikolon (;) unter Windows
- Der Punkt (.) ist wichtig für anonyme Pakete
- Das Verzeichnis, in welchem sich die JDK-Klassenbibliothek befindet, muss **nicht** im CLASSPATH vorhanden sein

CLASSPATH (3)

Beispiel:

- Im Verzeichnis `/languages/java` gibt es ein Paket namens `einaus` (Unterverzeichnis `einaus`), welches die Datei/Klasse `IO` enthält
- und ein Paket names `spiele.schach` (Unterverzeichnis `spiele/schach`), welches u.a. die Dateien/Klassen `SchachSpieler` und `SchachSpielzug` enthält.
- Sie wollen die Pakete zur Implementierung einer Klasse `MySchach` nutzen:

```
Datei: MySchach.java (im Verzeichnis /user/kai/java)
      import einaus.*;          → Zugriff auf Klasse IO
      import spiele.schach.*; → Zugriff auf SchachSpieler,…
      public class MySchach extends SchachSpieler { ... }

export CLASSPATH=.:/languages/java

Compilieren:  javac MySchach.java
Ausführen:    java MySchach
```

CLASSPATH (4)

Leicht modifiziertes Beispiel:

- dieselben Voraussetzungen, nur Sie wollen ein eigenes Paket definieren:

```
Datei: MySchach.java (im Verzeichnis /user/kai/java/kschach)
      package kschach;
      import einaus.*;          → Zugriff auf Klasse Terminal
      import spiele.schach.*; → Zugriff auf SchachSpieler, ...
      public class MySchach extends Schachspieler { ... }

export CLASSPATH=.:/languages/java:/user/kai/java
Compilieren:  javac MySchach.java
Ausführen:    java kschach.MySchach (von wo aus, ist egal!)
```

Vollständiger Klassenname

Achtung: alle Verzeichnisse und `class`-Dateien müssen lesbar sein !

jar

`jar:` Java-Hilfsprogramm zum Packen und Komprimieren von Java-Dateien

Entpacken: **`jar xvf hamstersimulator.jar`**

Packen: **`jar cvf simulator.jar *.class hamster\*.class`**

Executable jar: spezielle ausführbare jar-Datei (Doppelklick) mit Informationen über die aufzurufende main-Funktion

Umgang mit Paketen

Demo

Pakete in Eclipse

Demo

Zusammenfassung

- Java-Pakete: Hilfsmittel zur Strukturierung von Klassenbibliotheken
- Klassenbibliothek: Sammlung von nützlichen, häufig gebrauchten Klassen, die (anderen) Programmierern zur Verfügung gestellt werden
- CLASSPATH: In den CLASSPATH müssen die Verzeichnisse (bzw. zip- oder jar-Dateien) aufgenommen werden, in denen der Java-Compiler und -Interpreter nach Paketen suchen soll
- jar: Java-Hilfsprogramm zum Packen und Komprimieren von Java-Dateien (insbesondere Klassenbibliotheken)