

ESOP - Pakete & Generics

Assoc. Prof. Dr. Mathias Lux
ITEC / AAU

Quellen / Sources



- Mössenböck (2014) Sprechen Sie Java



- The Java Tutorials:
 - Packages:
<http://docs.oracle.com/javase/tutorial/java/package/index.html>
 - Generics:
<https://docs.oracle.com/javase/tutorial/extra/generics/index.html>
- Sierrag & Bates (2005) Head First Java
 - Packages: p.155+, p.587+
 - Generics: p. 540+, ...

Pakete



Sammlung zusammengehöriger Klassen

- Bring mehr Ordnung in Programme
- Vermeidung von Namenskonflikten
- bessere Zugriffskontrolle



Beispiele für Pakete



- `java.lang`
 - `System`, `String`, `Integer`, `Character`, `Object`, `Math`, ...
- `java.io`
 - `File`, `InputStream`, `OutputStream`, `Reader`, `Writer`, ...
- `java.awt`
 - `Button`, `CheckBox`, `Frame`, `Color`, `Cursor`, `Event`, ...
- `java.util`
 - `ArrayList`, `Hashtable`, `BitSet`, `Stack`, `Vector`, ...

Anlegen von Paketen



- Anweisung in der ersten Zeile der Datei
- Fehlt die package-Zeile dann gehört die Klasse zu namenlosen Standardpaket.

Datei *Circle.java*

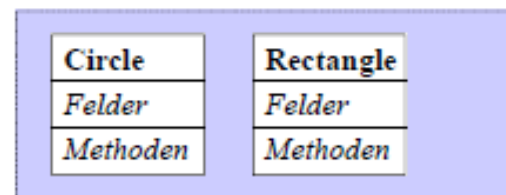
```
package graphics;  
class Circle {  
    ...  
}
```

Datei *Rectangle.java*

```
package graphics;  
class Rectangle {  
    ...  
}
```

← 1. Zeile der Datei

Paket *graphics* enthält die Klassen *Circle* und *Rectangle*

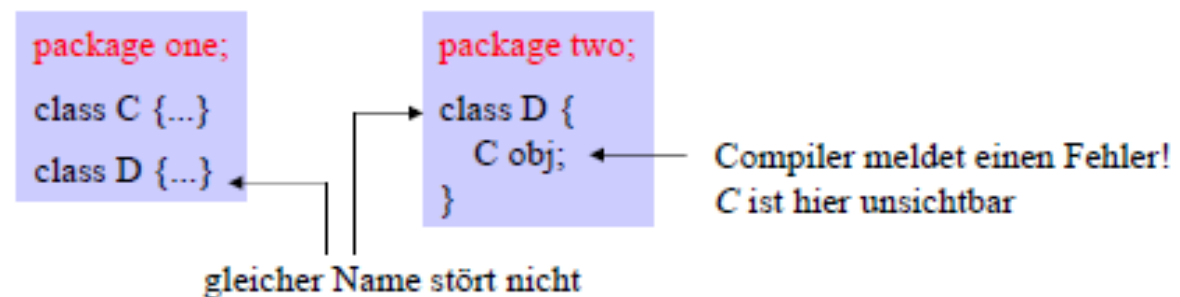


Paket *graphics*

Pakete als Sichtbarkeitsgrenzen



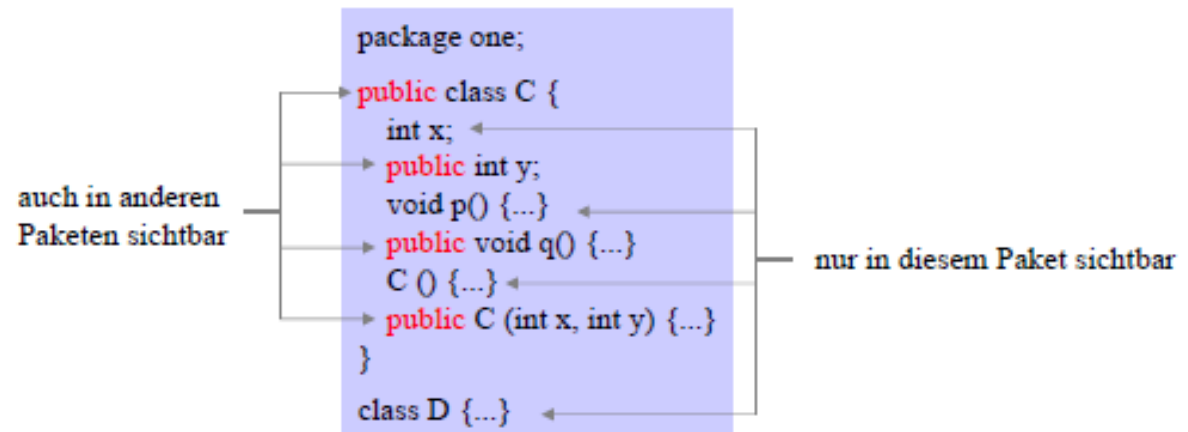
- Was in einem Paket deklariert ist, ist in einem anderen unsichtbar.
 - Verschiedenen Paketen & gleiche Namen möglich
 - Name muss nicht global eindeutig sein.



Export von Namen



- Namen können mit dem Zusatz `public` exportiert werden
 - sie sind dann in anderen Paketen sichtbar
 - `public`-Felder und -Methoden werden nur dann exportiert, wenn die Klasse selbst `public` ist.
 - lokale Variablen und Parameter können nicht exportiert werden.



Import von Klassennamen



- Exportierte Klassennamen können in anderen Paketen importiert werden
 - Durch gezielten Import der Klasse
 - Import aller `public`-Klassen eines Pakets
 - Qualifikation mit dem Paketnamen

```
package myPack;  
import graphics.Circle;  
import one.C;  
class MyClass {  
    Circle c;  
    ...  
}
```

```
package myPack;  
import graphics.*;  
class MyClass {  
    Circle c;  
    Rectangle r;  
    ...  
}
```

```
package myPack;  
class MyClass {  
    graphics.Circle c1;  
    java.awt.Circle c2;  
    ...  
}
```

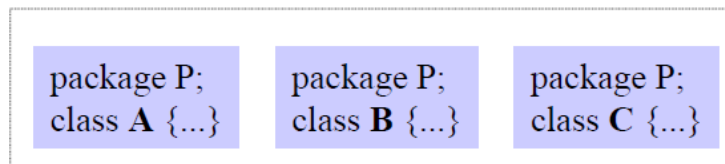

Pakete und Verzeichnisse



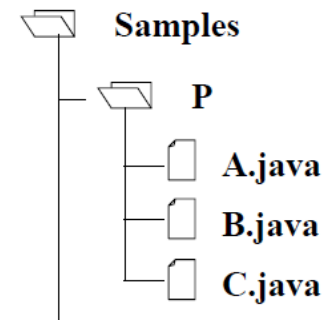
Pakete werden auf Verzeichnisse abgebildet, Klassen auf Dateien

Klasse C \Rightarrow Datei C.java

Paket P \Rightarrow Verzeichnis P



Paket P



Übersetzung und Ausführung mit dem JDK

```
cd C:\Samples
```

```
javac P/A.java
```

```
java P/A } beides möglich
java P.A }
```

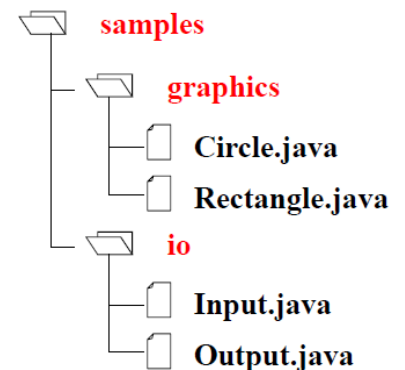
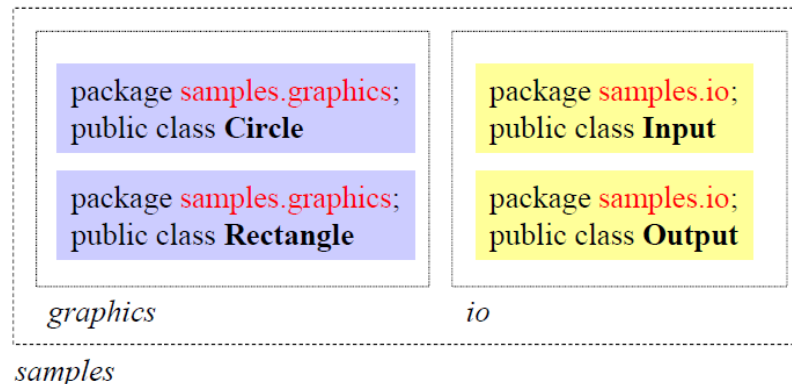
Geschachtelte Pakete



- Pakete können zu größeren Paketen zusammengefasst werden.

```
import samples.graphics.Circle;           // importiert die Klasse Circle
import samples.graphics.*                 // importiert alle public-Klassen
                                         // aus samples.graphics

import samples.*;                         // importiert alle public-Klassen aus
                                         // samples (nicht aus samples.graphics)
```



Information Hiding (Erweiterung)



Sichtbarkeitsattribute für Felder und Methoden

```
private    int a; // nur in Klasse sichtbar, in der Element deklariert wurde
           int b; // nur im Paket sichtbar, in dem Element deklariert wurde
protected int c; // sichtbar:
               // - in der deklarierenden Klasse
               // - in deren Unterklassen (auch in anderen Paketen!)
               // - im deklarierenden Paket
public     int d; // auch in anderen Paketen sichtbar, wenn importiert
```



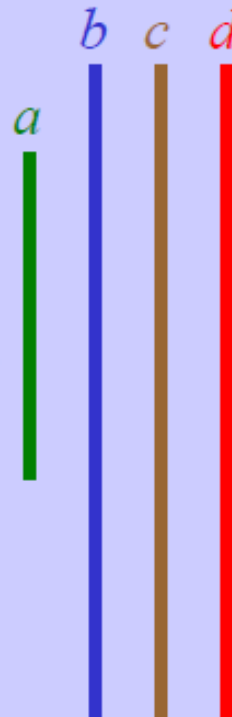
Sichtbarkeitsattribute



```
package one;

public class C {
    private int a;
    int b;
    protected int c;
    public int d;
}


public class D {
    ...
}
```



```
package two;
import one.C;

public class E extends C {
    C x = new C();
    x.a = ...;
    x.b = ...;
    x.c = ...;
    x.d = ...;
}

public class F {
    ...
}
```



Meet the ArrayList ...



Array

- Fixe Größe
- Einfügen via Index
- Funktioniert mit Basisdatentypen und Referenzdatentypen

ArrayList

- Dynamische Datenstruktur
- Funktioniert nur mit Referenzdatentypen
- `add(Object elem)`
- `remove(Object elem)`
- `contains(Object elem)`
- `isEmpty()`
- `get(int index)`
- `indexOf(Object elem)`

Read name file with ArrayList!

Generics



- Vorgehensweise bei Collections:

```
List list = new ArrayList();  
list.add("hello");  
String s = (String) list.get(0);
```

- Besser mit Generics

```
List<String> list = new ArrayList<String>();  
list.add("hello");  
String s = list.get(0);    // no cast
```

Nutzung von Generics



```
public class Box<T> {  
    // T stands for "Type"  
    private T t;  
  
    public void set(T t) { this.t = t; }  
    public T get() { return t; }  
}
```

- Eigene Klassen mit Generics definieren
- T wird wie Klasse benutzt.

Diamond Operator



```
Box<String> myBox = new Box<>()
```



Diamond
Operator

- Der Diamant kann genutzt werden, wenn der Compiler den Typ implizieren kann

Mehr über Generics ...



- Java Tutorial Trail
 - <http://docs.oracle.com/javase/tutorial/java/generics/index.html>

Traversing Collections



- for-each

```
for (Object o : collection)
    System.out.println(o);
```

- Iterator

```
public interface Iterator<E> {
    boolean hasNext();
    E next();
    void remove(); //optional
}
```

Listenoperationen



- **sort**
 - sorts a List using a merge sort algorithm, which provides a fast, stable sort. (A *stable sort* is one that does not reorder equal elements.)
- **shuffle**
 - randomly permutes the elements in a List.
- **reverse**
 - reverses the order of the elements in a List.
- **rotate**
 - rotates all the elements in a List by a specified distance.
- **swap**
 - swaps the elements at specified positions in a List.
- **replaceAll**
 - replaces all occurrences of one specified value with another.
- **fill**
 - overwrites every element in a List with the specified value.
- **copy**
 - copies the source List into the destination List.
- **binarySearch**
 - searches for an element in an ordered List using the binary search algorithm.
- **indexOfSubList**
 - returns the index of the first sublist of one List that is equal to another.
- **lastIndexOfSubList**
 - returns the index of the last sublist of one List that is equal to another.

Maps



- HashMap
 - als wichtiger Vertreter und Beispiel
 - HashSet als Basis (schneller Zugriff)
 - keine zugesicherte Ordnung
- TreeMap
 - TreeSet als Basis -> geordnet
- LinkedHashMap
 - LinkedHashSet -> hash + geordnet

HashMap



- Beispiel
 - `HashMap<String, List<String>> student2course`