Pakete

Idee

Paket = Sammlung zusammengehöriger Klassen (Bibliothek).

Zweck

- mehr Ordnung in Programme bringen,
- bessere Kontrolle der Zugriffsrechte (wer darf auf was zugreifen),
- Vermeidung von Namenskonflikten.

Beispiele

Paket	*enthaltene Klassen*
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, Random, ...
...	...

Anlegen von Paketen

Datei *Circle. java*

```
package graphics;

class Circle {
   ...
}
```

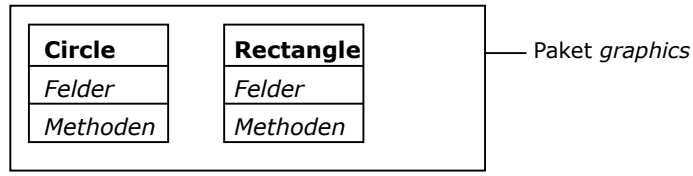
Datei *Rectangle. java*

```
package graphics;

class Rectangle {
   ...
}
```

← 1. Zeile der Datei

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



Wenn *package*- Zeile fehlt, gehören die Klassen zu einem namenlosen Standardpaket.

Pakete als Sichtbarkeitsgrenzen

Was in einem Paket deklariert ist, ist in anderen Paketen unsichtbar

```
package  one;

class C { ... }
class D {...  }
```

```
package two;

class D {
   C obj;
}
```

Compiler meldet einen Fehler !
C ist hier unsichtbar !

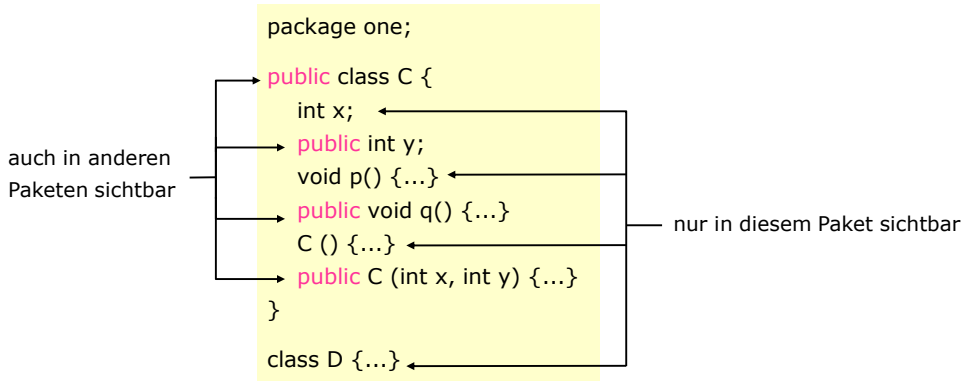
gleicher Name stört nicht

Zweck

- In verschiedenen Paketen können gleiche Namen verwendet werden.
- Schutz vor (unabsichtlicher) Zerstörung.

Pakete als Sichtbarkeitsgrenzen

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.

Pakete als Sichtbarkeitsgrenzen

Exportierte Klassennamen können in anderen Paketen importiert werden.

Durch gezielten Import der Klasse

```
package myPack;

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

Durch Qualifikation mit dem Paketnamen

```
package myPack;

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

Durch Import **aller** public- Klassen eines Pakets.

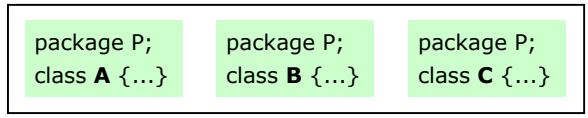
```
package myPack;

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

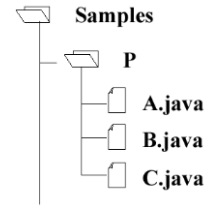
Pakete und Verzeichnisse

Pakete werden auf Verzeichnisse abgebildet, Klassen auf Dateien.

Klasse C	Datei C. java
Paket P	Verzeichnis P



Paket P



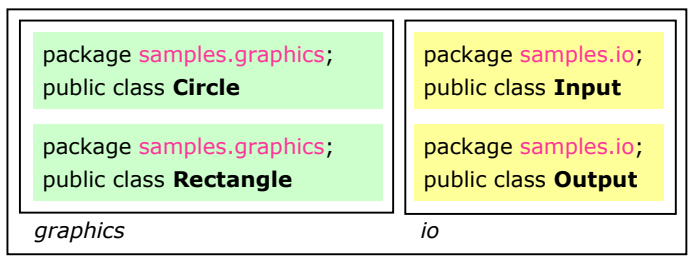
Übersetzung und Ausführung mit dem JDK.

```
cd C:Samples
javac  P/A.java
java   P/A
java   P.A
```

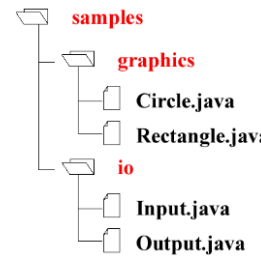
} beides möglich

Geschachtelte Pakete

Pakete können zu größeren Paketen zusammengefaßt werden.



samples



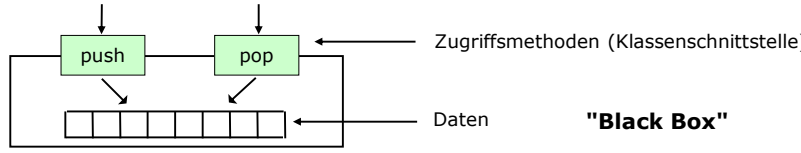
Benutzung

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)
samples.io.Output out;	Qualifikation einer Klasse aus einem geschachtelten Paket

Information Hiding (Geheimnisprinzip)

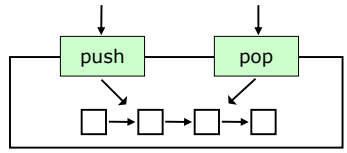
Prinzip

- Verstecke die Implementierung komplexer Datenstrukturen vor den Klienten.
- Erlaube den Zugriff auf die Daten nur über Methoden.



Warum?

- Verringert die Komplexität (Arbeiten mit den Daten wird einfacher).
- Implementierung der Daten kann geändert werden, ohne daß Klienten etwas merken.
- Schutz vor mutwilliger oder unabsichtlicher Zerstörung.



Sichtbarkeitsattribute

für Felder und Methoden

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

```
package one;

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

a b c d

```
package two;

import one.C;
public class E extends C {
...
}
public class F {
...
}
```

c d

Beispiel: Stack mit Information Hiding

```
public class Stack {
   private int[] data;
   private top;
   public Stack (int size) {
       data = new int[ size]; top = -1;
   }
   public void push (int x) {
      if (top >= data. length) error(" stack overflow");
      else data[++ top] = x;
   }
   public int pop () {
      if (top < 0) error(" stack underflow");
      else return data[ top--];
   }
   private void error (String msg) {
      Out. println( msg);
      System. exit( 0);
   }
}
```

Klassenschnittstelle

Stack
Stack() push(int x) pop(): int

Dokumentationskommentare

Dokumentationskommentare

/** ... */

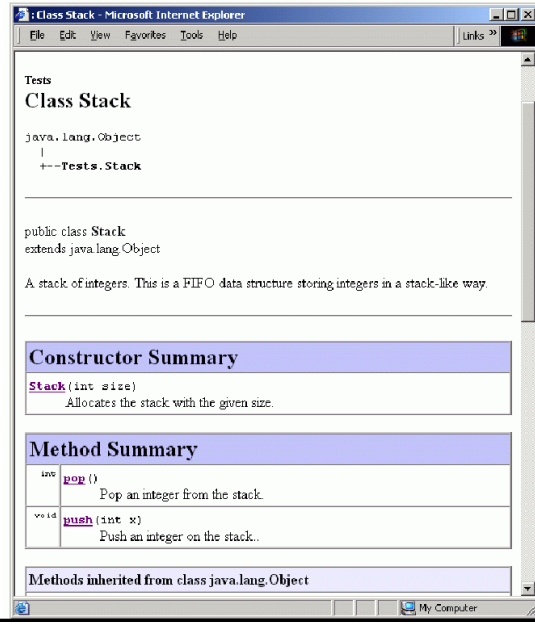
können vor die Deklarationen von Klassen, Methoden, Feldern gesetzt werden.
Werkzeug javadoc erzeugt daraus Dokumentation in HTML.

```
/** A stack of integers.
    This is a FIFO data structure storing integers in a stack- like way. */
public class Stack {
    /** The elements in the stack. */
    private int[] data;
    ...
   /** Push an integer on the stack.
        If the stack is full an error is reported and the program stops. */
   public void push (int x) {
   ...
   }
   ...
}
```

1. Satz bis Punkt wird in Kurzdoku übernommen

Rest wird in Langdoku übernommen

Erzeugte HTML- Datei (Stack. html)



javadoc

Aufrufsyntax

javadoc [options] {filename | packagename}

Optionen

- -public zeigt nur public- Deklarationen
- -private zeigt public- und private- Deklarationen
- -d path gibt Zugriffspfad für mehrere Klassen und Pakete an
- ...

Beispiel

javadoc -public myDirectory/Stack.java

erzeugt:
myDirectory/Stack.html
diverse andere Übersichtsdateien

Vollständige Dokumentation von javadoc

http:// java. sun. com/ j2se/ javadoc/

Erzeugte HTML- Datei (Stack. html)