

Inf-Prog

08.02.2010

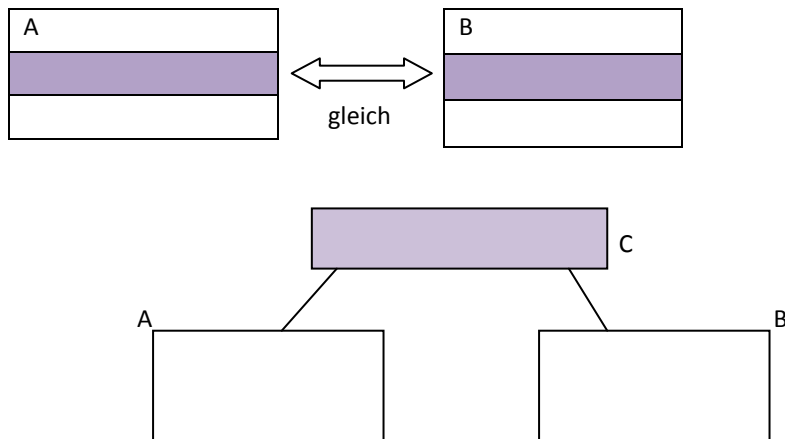
INHALT

Abstrakte Klassen, Schnittstellen, Pakete (5.6)	2
Abstrakte Klassen	2
Sonderfall: Schnittstellen	3
Beispiel: Schnittstellen für Tabellen	3
Pakete	3
Weitere Aspekte von Java (5.7)	4
Generizität (Java 5)	4
Threads	4
Remote Method Invocation (RMI)	4
Zusammenfassung der Vorlesung	4

ABSTRAKTE KLASSEN, SCHNITTSTELLEN, PAKETE (5.6)

ABSTRAKTE KLASSEN

- Klassen, von denen man keine Instanzen (Objekte) bilden kann
- einige Methoden nicht implementiert
- dienen zur Strukturierung: fasse Gemeinsamkeiten mehrerer Klassen in einer gemeinsamen Oberklasse zusammen.



Beispiel: Abstrakte Klasse für Benchmark

Konkreter Benchmark unbekannt, aber Wiederholung mit Zeitmessung möglich:

```

abstract class Benchmark {
    abstract void benchmark(); //Methode ohne Implementierung ≈ ohne Rumpf
    public long repeat(int count) { //Führe benchmark() count-mal durch
        long start = System.currentTimeMillis();
        for (int i = 0; i < count; i++) benchmark();
        return (System.currentTimeMillis() - start);
    }
}

```

Beachte: `new Benchmark()`; nicht möglich!

Konkreter Benchmark: Unterklasse von Benchmark + Implementierung von `benchmark()`

```

class MyBenchmark extends Benchmark {
    void benchmark() {...}
    public static void main(...) {... new MyBenchmark().repeat(1000) ...}
}

```

SONDERFALL: SCHNITTSTELLEN

- Abstrakte Klasse, wobei alle Methoden (implizit) abstrakt und alle Attribute (implizit) final static (d.h. Konstanten) sind.
- ⇒ Schnittstellen implementieren nichts.
- dienen zur Strukturierung und Dokumentation: Was implementiert eine Klasse?
- eine Klasse kann beliebig viele Schnittstellen implementieren (nicht aber beliebig viele Oberklassen haben).

BEISPIEL: SCHNITTSTELLEN FÜR TABELLEN

```
interface Lookup { //statt abstract class
    Object lookup(String name); //Liefert beliebiges Objekt zurück.
}

interface Insert {
    void insert(String name, Object value);
}

class MyTable implements Lookup, Insert {
    ... /konkrete Implementierung von lookup(...) und insert(..., ...)
}
```

Schnittstellen können wie abstrakte Klassen benutzt werden:

```
... void processValues(String[] names, Lookup table) {
    ...table.lookup(names[i]) ...
}
```

PAKETE

- Strukturierung von Klassensammlungen zu größeren Einheiten
- Jede Klasse gehört zu genau einem Paket: Angabe des Namens zu Programmbeginn (vor einer Klasse)

```
package mh.lehre.info1; //Hierarchische Paketnamen ≈ Dateisystem
class ...
```

- Jede Klasse in einem Paket kann jetzt über den Paketnamen angesprochen werden (Namenskonfliktvermeidung)

java.util.Date now = new java.util.Date();

{
}

Paketname Klasse in Paket

- Import kompletter Pakete: mache Paketklassennamen sichtbar

```
import java.util.Date

Date now = new Date() //oder import java.util.* ≈ alle Klassennamen in java.util

//nur möglich, falls keine andere Klasse "Date" aus anderen Paketen importiert wurde.
```

- Viele Pakete sind in Java API (Application Programmer Interface) vorhanden, z.B.
 - o java.util

- java.awt (Graphik-/Fensterprogrammierung)
- java.net (Netzwerkprogrammierung)

WEITERE ASPEKTE VON JAVA (5.7)

GENERIZITÄT (JAVA 5)

Klassen können Typparameter enthalten

Beispiel: Liste, wobei alle Elemente der Liste vom gleichen Typ T

```
class List<T> {
    T first;
    List<T> rest;
}

//Definition einer Studierendenliste (Klasse Student schon vorhanden)
List<Student> slist;
slist = new List<Student>();
slist.first = new Student();
```

- ⇒ mehr Typsicherheit, kein Vermischen unterschiedlicher Daten.
- ⇒ Benutzung bei Objektsammlungen (Felder, Listen, Bäume, Mengen, Tabellen, ...)

THREADS

Einfache Prozesse zur nebenläufigen Programmierung mit Synchronisationsmöglichkeiten (Objekte mit Semaphore)

REMOTE METHOD INVOCATION (RMI)

Verteilte Programmierung (mehrere Rechner):

Aufrufe von Methoden von Objekten, die auf anderen Rechnern liegen

- ⇒ Modul „Fortgeschrittene Programmierung“

ZUSAMMENFASSUNG DER VORLESUNG

Wichtige Aspekte der Programmierung:

- Entwickle gute Programmstruktur
- Schichtenentwurf von Programmen: unterstützt durch Module, Objekte, Pakete
- Strukturierung durch Bildung von Abstraktionen
 - Funktionen/Prozeduren: Abstraktion konkreter Verfahren
 - Abstrakte Datentypen: Abstraktion konkreter Darstellungen
 - Objekte: Darstellung + Verfahren abstrahieren.
- Vermeide unnötige Komplexitäten im Verständnis von Programmen
 - keine globalen Variablen verändern, falls nicht unbedingt notwendig.
 - keine Funktionen mit Seiteneffekten (Abhängigkeit der Auswertungsreihenfolge) sondern Prozeduren mit fester Reihenfolge (begin...)
- Keine iterative Verfahren, wenn rekursive Verfahren verständlich/ausreichend sind. (insbesondere: rekursive Datenstrukturen (Listen, Bäume) => rekursive Verfahren)

Diese Programmierprinzipien gelten für alle höheren Programmiersprachen