

Dies ist keine „Musterlösung“, sondern eine gute von vielen möglichen Lösungen. Kommentare, die nicht Teil der Lösung sind, sind kursiv gesetzt.

Aufgabe 7-1:

1.a. Die Signatur gibt nur die Parameter, Rückgabe und deren Typen einer Schnittstelle an. Zur Schnittstelle gehören auch die Verwendung (Vorbedingungen und Zusicherungen), d.h. das „was“.

1.b. Eine Komponente ist ein Modul, d.h. ein funktionales, wieder verwendbares, nutzbares Teilsystem, während eine Klasse meist alleine keinen Sinn macht. In ganz seltenen Fällen besteht eine Komponente nur aus einer Klasse oder nur aus einer Methode, meistens ist sie umfangreicher.

1.c. *Kohäsion* beschreibt den Grad des Zusammenhalts der Teile eines Moduls. Der Zusammenhalt definiert sich aus der Anzahl der Benutzungen/Aufrufe der Teile untereinander. Eine hohe Kohäsion spricht für einen hohen Grad an Gemeinsamkeit (das sog. Geheimnis, *concern* oder Entscheidung) der Teile. Die *Kopplung* ist die Außenbeziehung des Moduls zu anderen Modulen, also die Nutzung der Module untereinander. Eine niedrige Kopplung spricht für einen hohen Wiederverwendungsgrad. Eine Modularisierung ist – vereinfacht gesagt – dann gut, wenn die Kohäsion pro Modul hoch und die Kopplung der Module niedrig ist.

2.

Schichten: KVV mit den Schichten Datenbank, Serverlogik und Oberfläche. Oder die Telekommunikationsschichten von ISO/OSI.

Datenflussnetze: Reportgenerator, z.B. zum Ausdrucken von Serien-E-mails: Adressdatenaufbereitung -> Abfragen fehlender Werte -> Briefgenerierung -> Abschicken.

Objektnetze: Steckt überall drin...

Ereignissteuerung: In Benutzungsoberflächen, z.B. auf Basis von Swing/AWT

Ablagebasierte Struktur: Multi User Dungeons (MUD) und alle Datenbank Anwendungen

Interpretierer: Excel, wo ständig die Formeln geparkt und daraufhin neue Werte generiert werden.

Ein Softwaresystem kann viele solcher Stile enthalten. Sie ergänzen sich. Man kann bei jedem gegebenen Beispiel also nachfragen, welche Stile ansonsten noch dort drin stecken.

3.a. Echtzeit => Ereignisbasiert (z.B. Insulinpumpe). *Eigentlich ist hier Unterbrechungsbasiert angesagt, da ein Ereignis meist sofort bearbeitet werden muss, d.h. es wird der aktuelle Lauf unterbrochen.*

3.b. Portabilität => Schichtenbasiert (z.B. alle Java-Anwendungen: Die Java-Engine ist eine Schicht über dem Betriebssystem), weil sich die Schichten austauschen lassen.

3.c. Speicher => Objektnetz (z.B. alle eingebetteten Systeme, weil die Objektnetze beliebig optimiert werden können und somit weniger Schnittstellenballast haben.

Aufgabe 7-2:

1. Bis „Eine solche Volltextsuche...“ ist es Anforderung, die Trennung Indizieren/Suchen ist Entwurf und der letzte Satz ist beides: Die Anforderung ist es „in verschiedenen Semestern getrennt suchen zu können“; Entwurfsentscheidung ist es, deshalb „mehrere Indizes verwalten zu können“.

2. Gemeint sind nicht Java-Interfaces, siehe 1.a.

Methodensignaturen sind o.k., aber man muss auch dazu schreiben, was die Methoden machen. Es folgt eine simple Lösung. Es ist nicht zwingend nötig, hier mit Vorbedingungen und Zusicherungen zu kommen – wäre natürlich schön!

```
// Erzeugt leeren Index, der mit search() durchsucht werden kann
Index createIndex ()
```

```
// Löscht einen Index. i wurde mit createIndex() erzeugt
void deleteIndex (Index i)
```

```
// Fügt ein neues Dokument zu einem Index hinzu. Der Index kann
// leer sein, aber nicht null. Zurückgegeben wird eine ID, die
// das indizierte Dokument referenziert. Diese ID wird bei
// search() zurückgeliefert.
DocumentID addDocumentToIndex (Document d, Index i)
```

```
// Liefert alle DocumentIDs des Index i zurück, in deren
// zugehörigem Dokument d (siehe addDocumentToIndex) der Suchtext
// s vorkommt, vorausgesetzt, vorher wurde
// addDocumentToIndex(d,i) aufgerufen.
DocumentIDList search (String s, Index i)
```

3. siehe Bild

