

Aufgabe 7-1*: (Begriffe)

1. Erläutern Sie den Unterschied oder Zusammenhang zwischen...

- a. Schnittstelle und Signatur
- b. Klasse und Komponente
- c. Kohäsion und Kopplung

2. Nennen Sie zu jedem der in der Vorlesung genannten Architekturstile ein Ihnen bekanntes Softwaresystem, das diesen Stil verwendet. Woher wissen Sie dies jeweils oder woran erkennt man es jeweils? (Nehmen Sie nicht die auf den Folien schon genannten Beispiele.)

3. Welche Architekturstile sind für welche der folgenden drei nicht-funktionalen Anforderungen besonders geeignet?

- a. Echtzeitverhalten (d.h. zugesicherte Reaktionszeiten des Systems)
- b. hohe Portabilität (über mehrere Betriebssystemplattformen)
- c. geringer Speicherplatzverbrauch

Aufgabe 7-2: (Modularisierung und Schnittstellen)

In unserem Prüfungsverwaltungssystem (siehe vergangene Übungsblätter) soll nun auch eine so genannte „Volltextsuche“ eingebaut werden:

Man soll einen Suchtext eingeben können, so dass als Suchergebnis daraufhin alle Veranstaltungen angezeigt werden, bei denen der Suchtext im Titel oder in der Beschreibung der Veranstaltung enthalten ist. Eine solche Volltextsuche wird in der Regel in zwei Verarbeitungsschritten realisiert:

- 1. Die zugrunde liegenden Datensätze werden zunächst in einer Art Schlagwortindex (ähnlich wie bei einem Buch) gespeichert, der einzelne Wörter auf die Veranstaltungen abbildet, in denen dieses Wort vorkommt.*
- 2. Auf Basis eines solchen Indexes kann dann effizient gesucht werden.*

Es sollen mehrere Indizes verwaltet werden können, um z.B. in verschiedenen Sprachen getrennt suchen zu können.

1. Welche Aussagen in diesem Text sind Anforderungsbeschreibungen und welche Aussagen sind Entwurfsentscheidungen?

2. Erstellen Sie die Schnittstelle eines *Volltextsuchers*, aus der die genannte Funktionalität hervorgeht. Bedenken Sie sämtliche oben genannten Anforderungen und Entwurfsentscheidungen. Sie sollten zudem eine möglichst wiederverwendbare, d.h. auch in anderen Anwendungen einsetzbare Komponentenschnittstelle entwerfen.

3. Überlegen Sie sich zwei Teilkomponenten, aus denen der *Volltextsucher* zusammengestellt werden kann. Erstellen Sie ein UML-Komponentendiagramm, das die gesamte Modularisierung verdeutlicht.

Aufgabe 7-3** (KWIC-Analyse und -Erweiterung)

** Diese Aufgabe kann als Bonus-Aufgabe von allen denen bearbeitet werden, die sich ein Plus verdienen möchten und ist bis zum Mittwoch 12.06. 16:00 abzugeben. Diese Aufgabe darf zu zweit bearbeitet werden, was insbesondere für Studierende im Nebenfach empfohlen wird, da ein paar Kenntnisse im Bereich nicht-sequentielle Programmierung benötigt werden.

Das in der Vorlesung vorgestellte KWIC-Programm liegt in allen drei dort genannten Versionen auf der Website: KWIC1, 2 und 3. Wir wollen zunächst die Implementationen analysieren und dann Version KWIC3 um eine simple Benutzungsoberfläche erweitern.

1. Schauen Sie sich KWIC1 an. Dort wurden auch Java *interfaces* eingesetzt. Welche Vorteile hat der Einsatz von *interfaces* (im Allgemeinen in Java und nicht nur im speziellen Fall von KWIC)? Vergessen Sie nicht, dass Java auch abstrakte Klassen kennt.

2. Betrachten Sie die Implementation (zu finden auf der Website) von KWIC3, der KWIC-Version mit *UString* und *WordList*. Lassen Sie KWIC3 laufen (`main()` ist in `kwic3.KWIC`) mit einem Parameter, der eine Textdatei mit den Buchtiteln beinhaltet (eine passende Datei ist anbei). Legen Sie einen Ausdruck ihrer Abgabe bei.

3. Derzeit kann KWIC nur aus einer Datei lesen. Wir wollen nun auch eine interaktive Eingabe anbieten. Unter `kwic3.impls.SimpleGui` ist ein simples Eingabefenster implementiert, mit dem man Textzeilen mittels Klick auf OK eine nach der anderen eingeben kann. Mit Klick auf END wird die Eingabe beendet und damit die Oberfläche geschlossen. *SimpleGui* speichert allerdings nichts ab. Es ist lediglich ein Modul, das zu ähnlichen Eingabezwecken wie bei KWIC benutzt werden kann.

a. Programmieren Sie für die Anbindung einer interaktiven Eingabe eine neue Klasse `kwic3.impls.GuiReader` als Unterklasse von *SimpleGui*, die `kwic3.interfaces.Reader` implementiert und als Fenster-basierter Reader für KWIC3 fungieren kann¹. Sie dürfen natürlich keine der vorgegebenen Klassen ändern! Gehen Sie dabei allein nach den dokumentierten Schnittstellen vor. Drucken Sie diese erste, kompilierbare Version von *GuiReader* aus.

b. Testen Sie nun Ihre Version. Dazu müssen Sie `KWIC.java` anpassen, so dass statt des *FileReaders* ein *GuiReader* erzeugt wird – dies ist die einzig erlaubte Änderung an den gegebenen Implementierungen. Funktioniert schon Ihre erste Version aus a.?

c. Wenn die erste Version nicht funktionierte, bedenken Sie bitte folgendes: Die Eingaben im Fenster laufen asynchron zu den Aufrufen von `Reader#getNextTitle()`. Nutzen Sie also Ihr Wissen über nichtsequentielle Programmierung². Ändern Sie das Programm, so dass es funktioniert und drucken Sie auch diese Version aus.

4. War die Schnittstelle zu `Reader#getNextTitle()` gut? Warum bzw. warum nicht?

¹ Beachten Sie dass KWIC die einzelnen Zeilen nicht immer schneller bearbeiten kann, als der Benutzer sie eingeben kann.

² Dieses haben Sie entweder aus der Vorlesung ALP4 oder aus Ihrer Beschäftigung mit Javas `wait()`, `notify()`, `synchronized`, etc. kennen gelernt. Dies sind wichtige und mächtige Java-Konstrukte.