Besprechung

Übung 1
Software Engineering

SE, 06.10.09

Dustin Wüest

Übungsabgaben

- **Im PDF**: vollständige Namen und Matrikelnummern
- **Name vom Zip-File / PDF**: Übungsnummer und Nachnamen

  z.B.: Ex1_Wueest_Schoen_Mueller

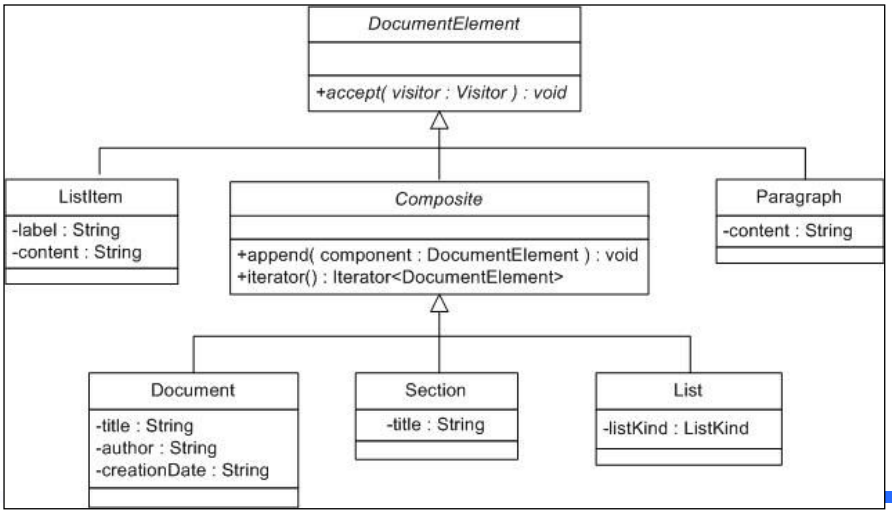
  (keine Umlaute / Sonderzeichen)
- Bei Nichteinhalten ist Punktabzug möglich
- Bei Abgabe kontrollieren, ob pdf / zip vollständig ist
- Wenn möglich 3er Gruppen
- Übungsabgaben auf Deutsch oder Englisch
- Modulbuchung bis Freitag, 17:00

Aufgabe 2.1 – UML Klassendiagramm

- Um Software besser zu verstehen / Überblick (auch Text)

  → Abstraktion und Layout!

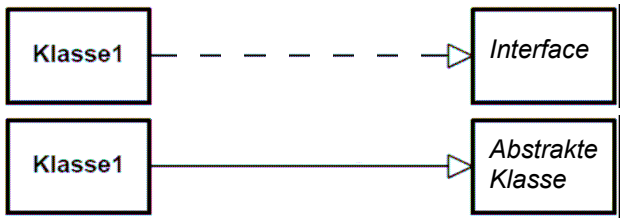
  → Konstruktoren, Getters/Setters, accept() Impl. weglassen



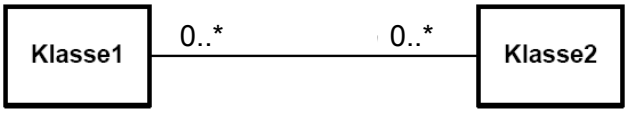
Aufgabe 2.1 – UML Klassendiagramm

- Vererbung

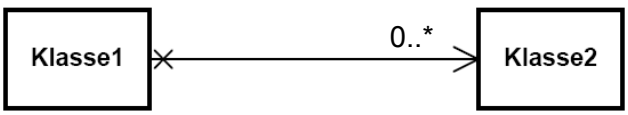
  (ohne Kardinalitäten)



- Assoziation

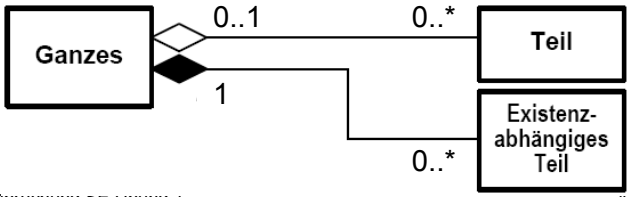


- Gerichtete Assoziation



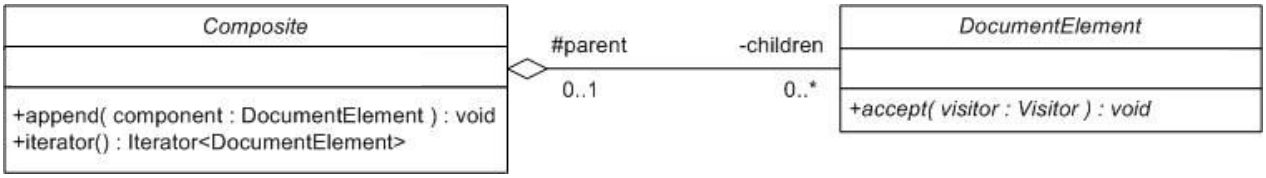
- Aggregation 

  Komposition 



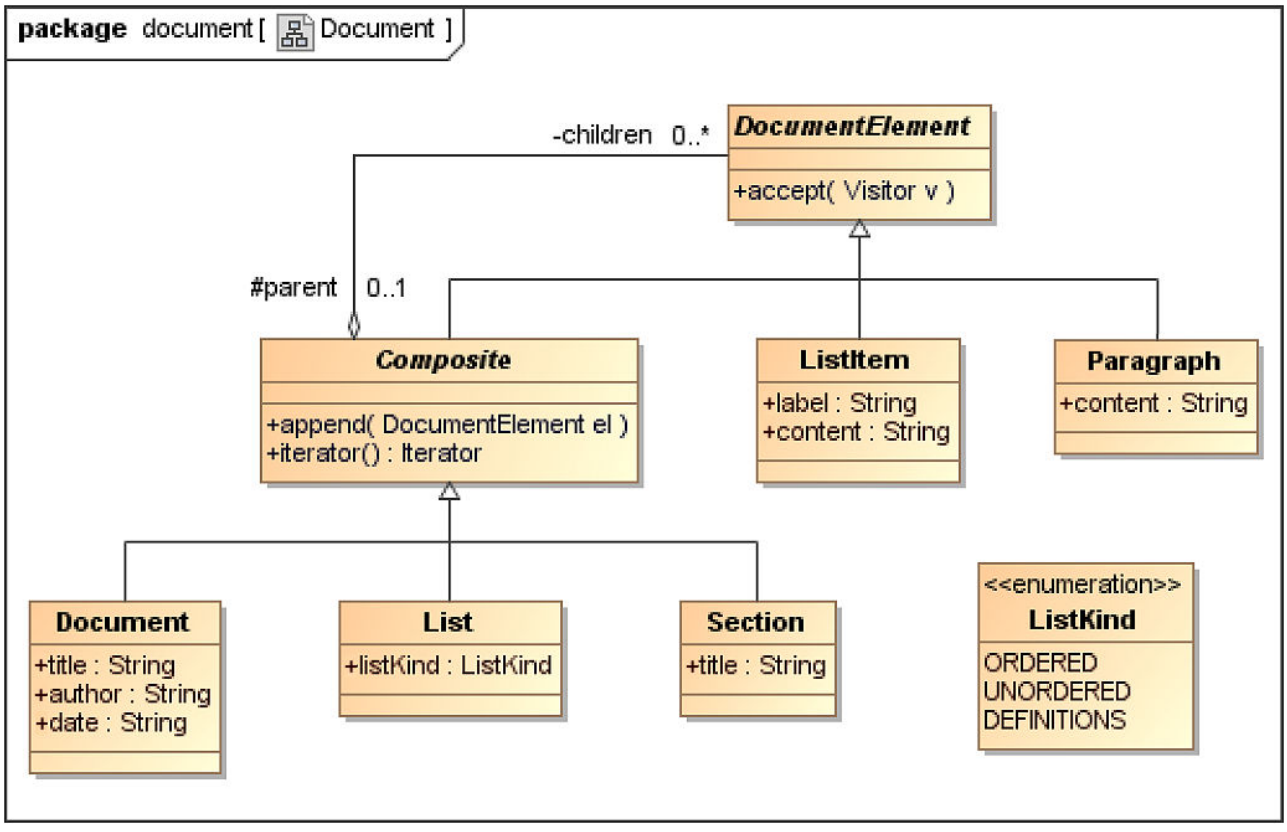
Aufgabe 2.1 – UML Klassendiagramm

- Assoziation im Modell → Attribut(e) im Code



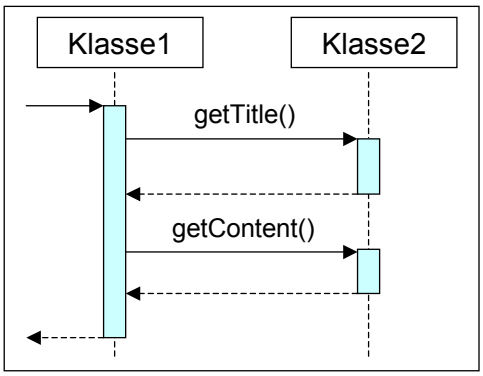
- Bidirektionale Assoziation (hier: Aggregation)
  → self.children.parent = self

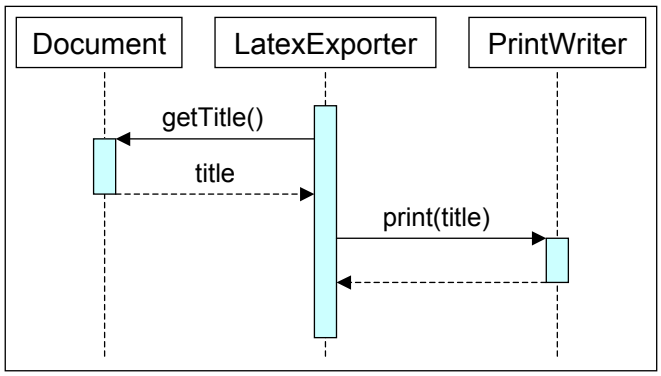
- Nicht (mehrere/verschiedene) Assoziationen zwischen den erbenden Klassen! Es ist **eine Assoziation zwischen den abstrakten Klassen** definiert. Die erbenden Klassen erben diese Eigenschaft.



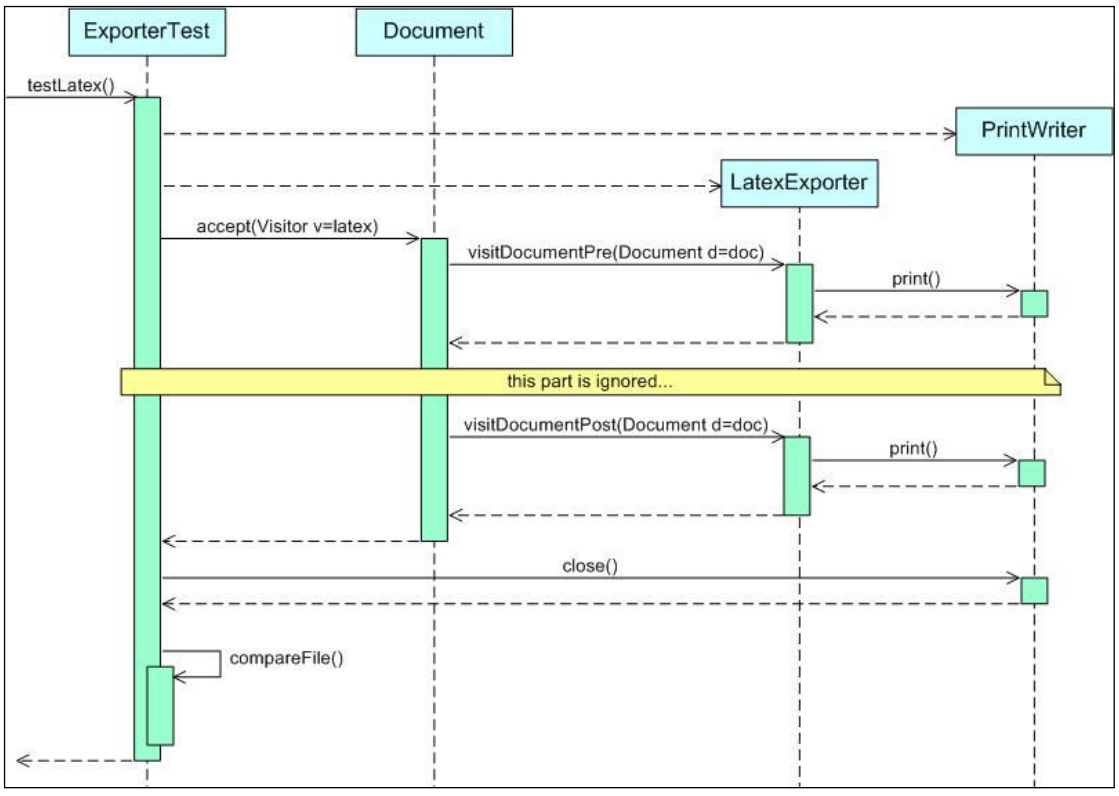
Aufgabe 2.2 – UML Sequenzdiagramm

- Aktivierungsbalken (bei laufenden Methoden)





- writer.println("..." + document.getTitle() + "...");
- Keine direkte Interaktion zwischen Document und PrintWriter



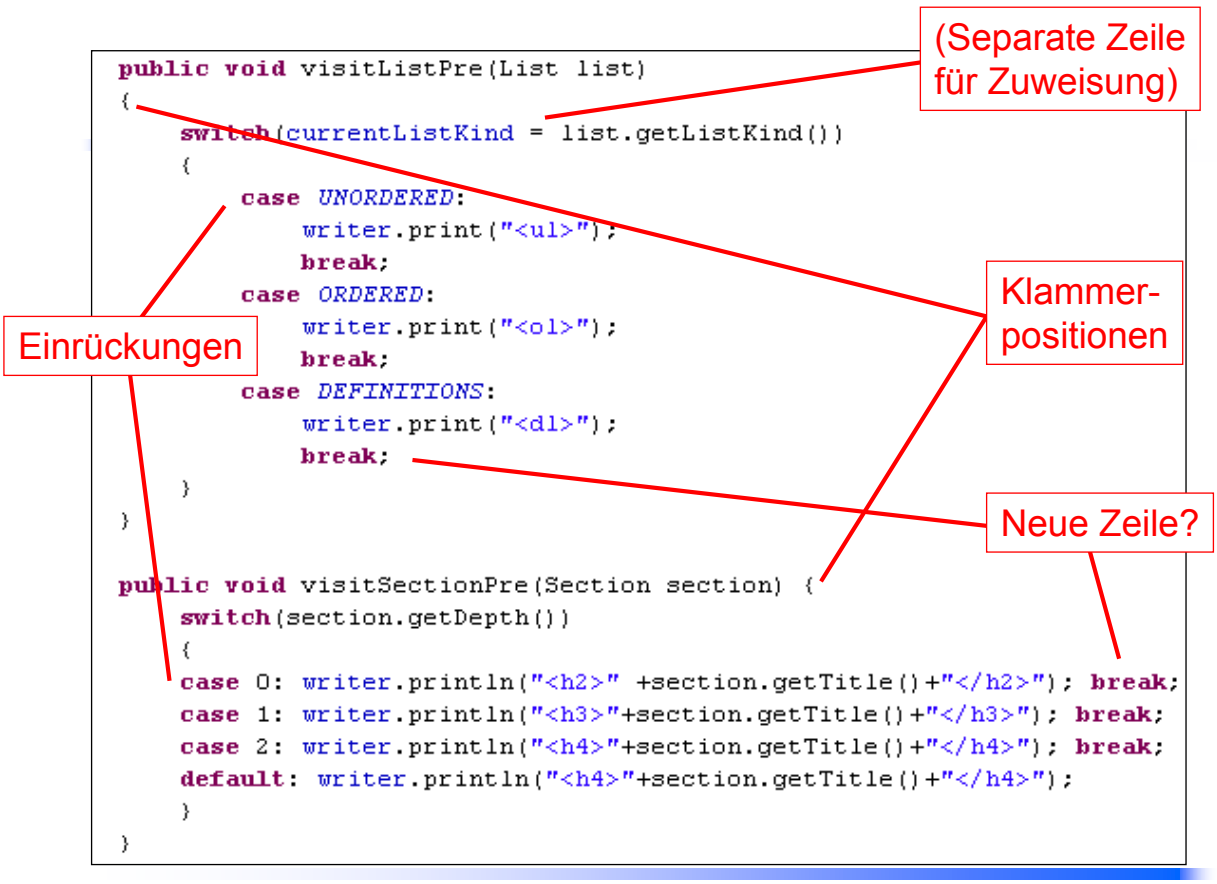
Aufgabe 2.3 – Wort, Zeichen, Abschnitte

- Wort und Zeichen:
  Definitionen, so wie sie im Dokument gebraucht werden
  Wort: Zeichenkette zwischen Leerzeichen (string.split(“ “))
  Zeichen: Charakters, Leerzeichen ausgenommen

- # Paragraphen:
  Word zählt alle Zeilenumbrüche (Paragraphenzeichen),
  ReportingSystem zählt die Paragraph Objekte

Aufgabe 3.1 - HTMLExporter

- Von den meisten gut gelöst (JUnit Test erfolgreich)
- **@Override** in Klassen, welche ein Interface implementieren, ab Java Version 6.0
- Auf einheitliche Formatierung achten (Klammern etc.)
  - Beispiel



Aufgabe 3.2 - Dokumentation

- Code kommentieren: JavaDoc /** ... */

  normaler Kommentar /* ... */ oder //

```
/**
 * Das ist JavaDoc. Damit es in den HTML Help Files
 * erscheint, muss es unmittelbar vor der dokumentierten
 * Methode/Klasse/Feld stehen
 */

/*
 * Normaler mehrzeiliger Kommentar. Für private Attribute
 * und Methoden; auch innerhalb von Methoden.
 */

// Normaler Kommentar, einzeilig.
```

Aufgabe 3.2 - Dokumentation

- JavaDoc: Blackbox → **was** macht die Klasse / die Methode
  (und nicht **wie**)
- JavaDoc Kopfkommentare von Methoden / Klassen:
  weder wann **noch** von wem eine Methode / eine Klasse aufgerufen wird
- Java Style Guide PDF (Kapitel 5)
  - Obligatorische Tags (*author, history, version, responsibilities* für Klassen und *pre, post, param, return* für Methoden)

Aufgabe 3.2 - Dokumentation

- Klassenkommentare und Attribute

```
package report.visitor.exporter;

import java.io.PrintWriter;


/**
 * This class exports a {@link report.document.Document} to an HTML file.
 * HTML tags conform to the underlying structure of the document elements.
 *
 * @author              Hans Muster
 * @history             2009-09-27 HM first version
 * @version             2009-09-27 HM 1.0
 * @responsibilities    This class generates an HTML file from a Document object
 */
public class HTMLExporter_Example implements Visitor {

    //Stores the enumerator type for the current list.
    private ListKind currentListKind;

    //Writes to the appropriate file
    private PrintWriter writer;
```

Aufgabe 3.2 - Dokumentation

- Methodenkommentare

```
/**
 * Writes the start tag of a list.
 *
 * @pre list != null
 * @post -
 * @param list the list which gets exported to HTML
 */
@Override
public void visitListPre(List list) {
    //The type of the list depends on the current list kind.
    currentListKind = list.getListKind();
    switch(currentListKind){
        case ORDERED: writer.println("<ol>"); break;
        case UNORDERED: writer.println("<ul>"); break;
        case DEFINITIONS: writer.println("<dl>"); break;
    }
}
```

Aufgabe 3.2 - Dokumentation

- **@pre**: was muss erfüllt sein, damit die Methode korrekt ausgeführt wird (ohne Exception)
- **@post**: Zustandszusicherung (ein Zustand, welcher sicher zutrifft, nachdem die Methode beendet ist)
- Z.B. für den Konstruktor:

  @pre writer != null

  @post writer is ready

Aufgabe 3.2 - Dokumentation

- Beispiel für @pre und @post: Pop()-Methode entfernt das oberste Element in einem Stack

```
//Die eingefügten Elemente werden in diesem Array gespeichert.
private Integer[] stackArray;

/**
 * Nimmt das oberste Element vom Stack und liefert
 * dessen Wert zurück.
 *
 * @pre stackArray != null && stackArray.length > 0
 * @post stackArray.length = stackArray.length @pre - 1
 * @return der Wert des entfernten Elements
 */
public int pop() {
    ...
}
```

Aufgabe 3.2 - Dokumentation

- JavaDoc Beispiel mit Eclipse

Übung 2 - Prototypen

- HTML
- Ohne HTML? → Links zwischen Seiten / Funktionen müssen klar nachvollziehbar sein

Allgemeines

Die nächsten Übungsstunden beginnen gemäss Abstimmung jeweils gleich nach der Vorlesung um 11:45 Uhr