

Besprechung

Übung 1 Software Engineering

SE, 19.10.10



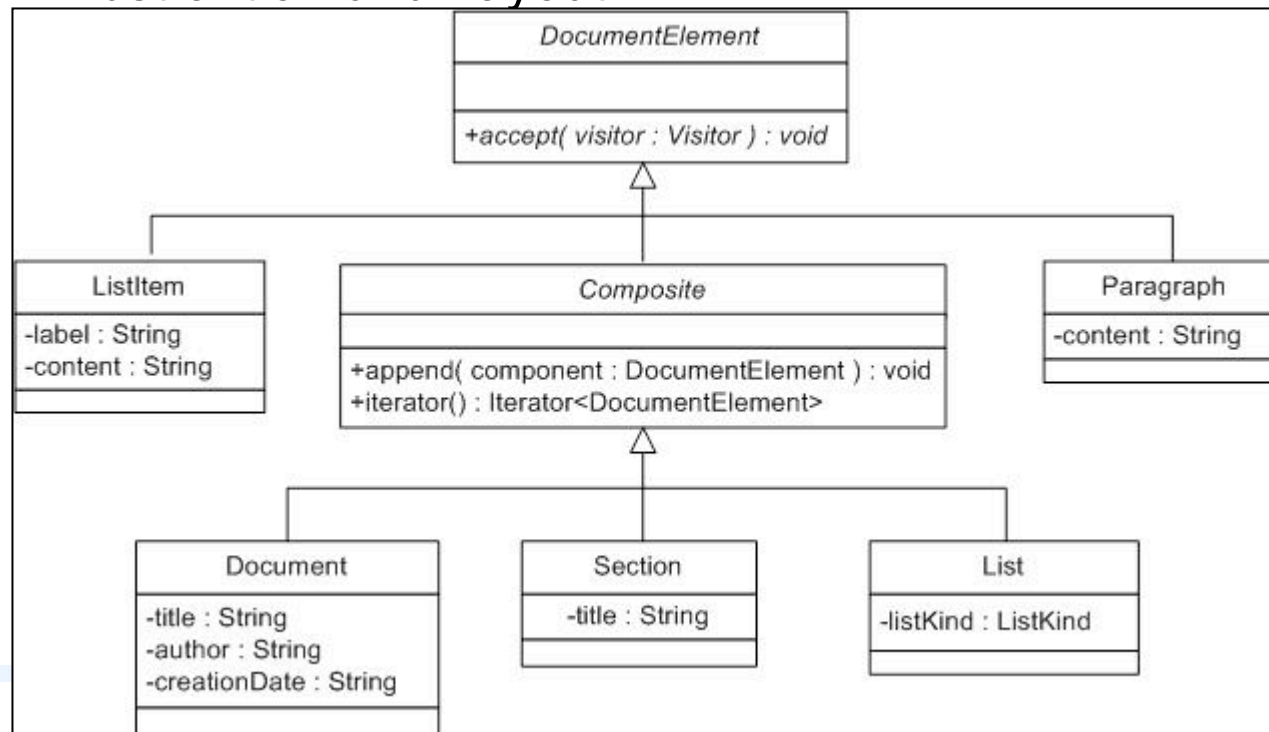
Dustin Wüest

Übungsabgaben

- **Im PDF:** vollständige Namen und Matrikelnummern
- **Name vom Zip-File / PDF:** Übungsnummer und Nachnamen
z.B.: Ex1_Wueest_Schoen_Mueller
(keine Umlaute / Sonderzeichen)
- E-Mail Betreff: **[SE EX HS10]** ...
- Bei Abgabe kontrollieren, ob pdf / zip vollständig ist
- Ausschliesslich 3er Gruppen
- Übungsabgaben auf Deutsch oder Englisch

Aufgabe 3.1 – UML Klassendiagramm

- Um Software besser zu verstehen / Überblick
 - Konstruktoren, Getters/Setters, Impl. Abstrakter Methoden weglassen
 - Abstraktion und Layout

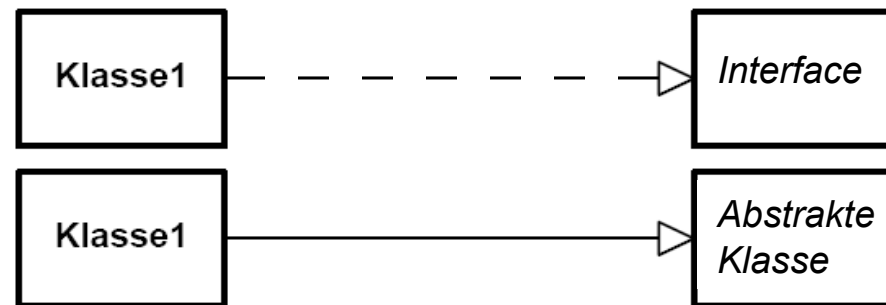


Layout
Beispiel

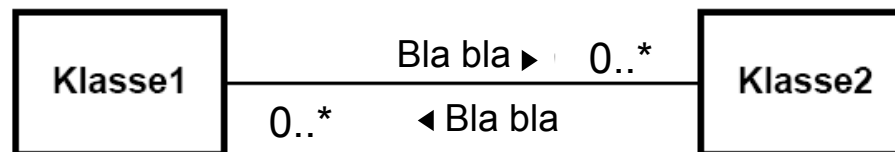
Aufgabe 3.1 – UML Klassendiagramm

- Vererbung**

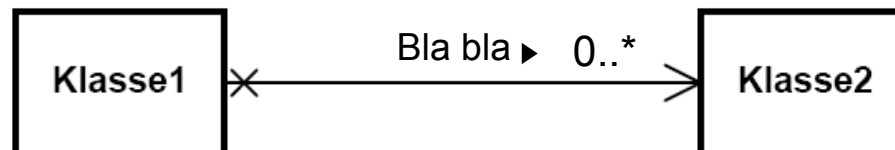
(ohne Kardinalitäten)



- Assoziation**

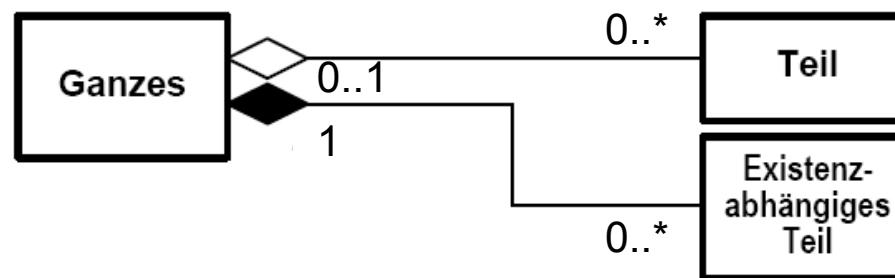


- Gerichtete Assoziation**

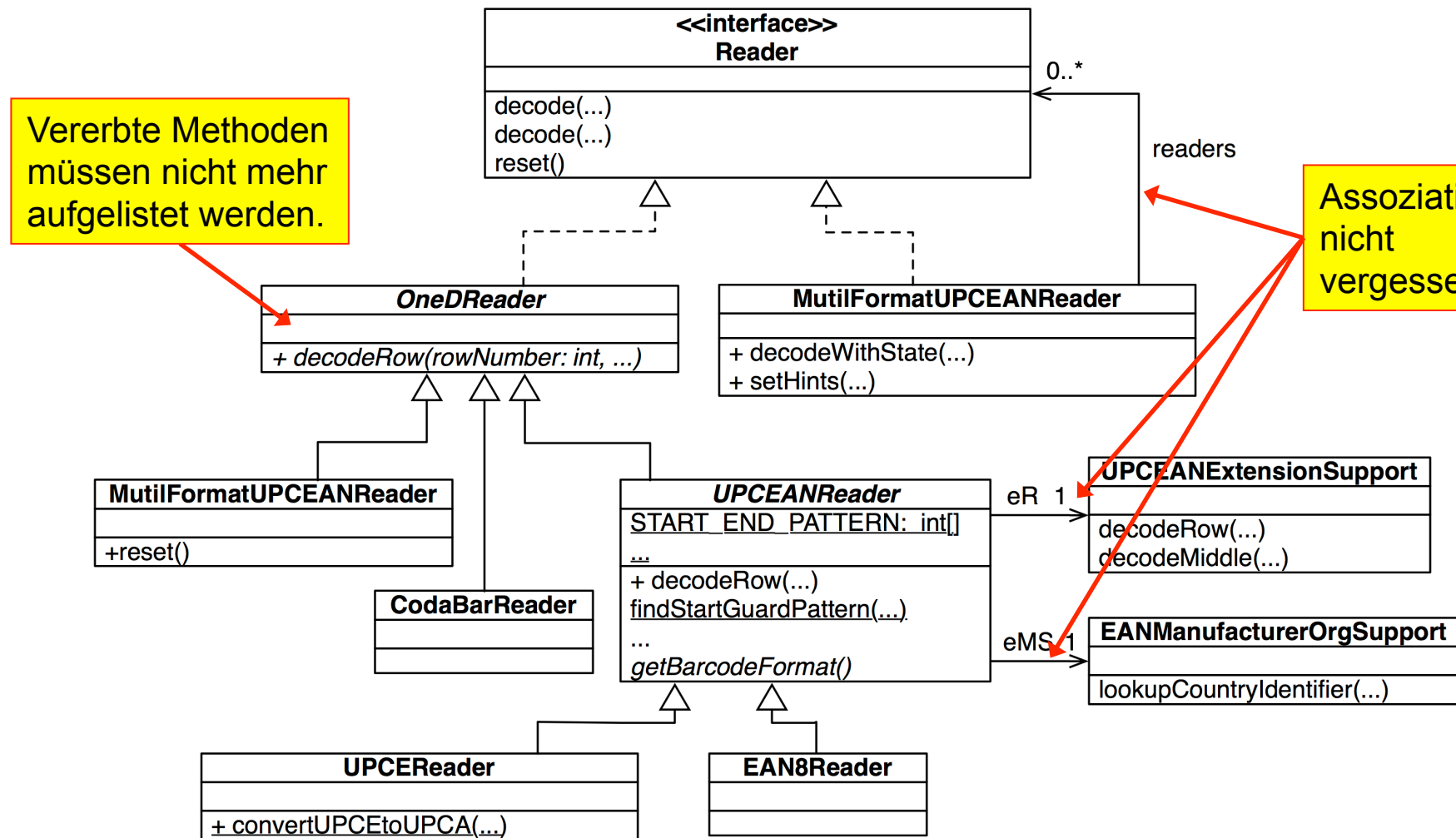


- Aggregation**

Komposition 



Aufgabe 3.1 – UML Klassendiagramm



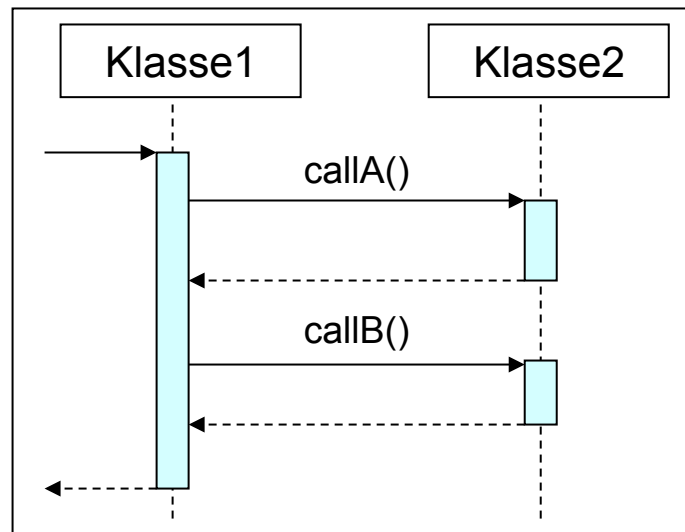
Aufgabe 3.2 – Arten von Strichcodes

- MultipleFormatWriter
 - Ist kein echter *Writer*
 - Wählt den korrekten *Writer* aus

Art	Lesen	Schreiben
Codabar	✓	✗
Code 39	✓	✓
Code 93	✓	✗
Code 128	✓	✓
EAN 8	✓	✓
EAN 13	✓	✓
ITF Code	✓	✓
UPC-A	✓	✗
UPC-E	✓	✗
Data Matrix	✓	✗
QR Code	✓	✓
RSS (GS1 DataBar) 14	✓	✗
RSS (GS1 DataBar) Extended	✓	✗
PDF417 Code	✓	✗

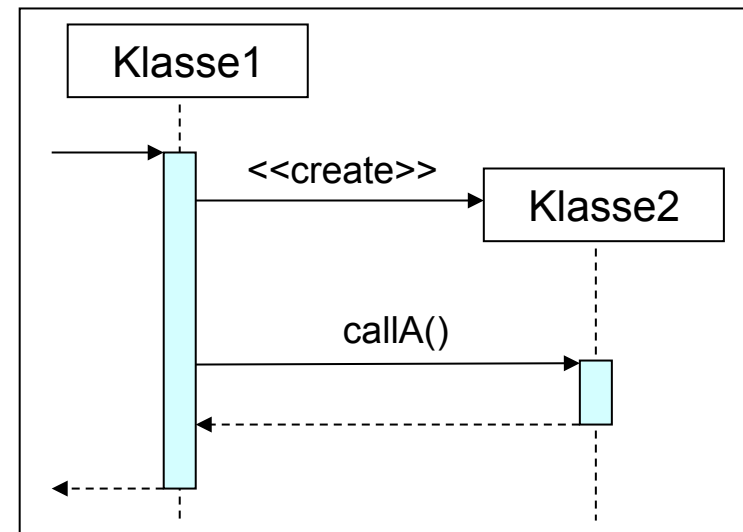
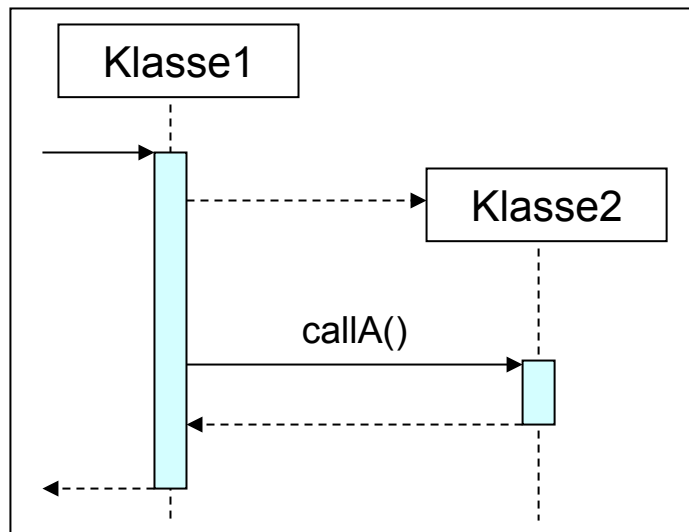
Aufgabe 3.3 – UML Sequenzdiagramm

- Aktivierungsbalken (bei laufenden Methode)
- Konstruktoraufruf
- Alternativen



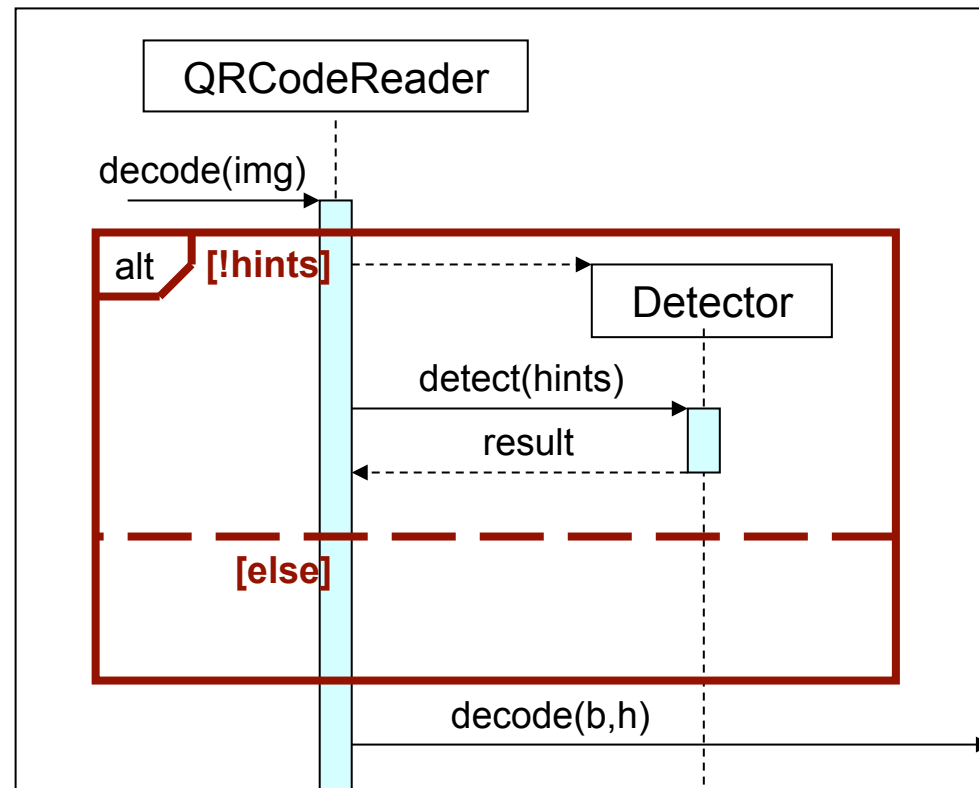
Aufgabe 3.3 – UML Sequenzdiagramm

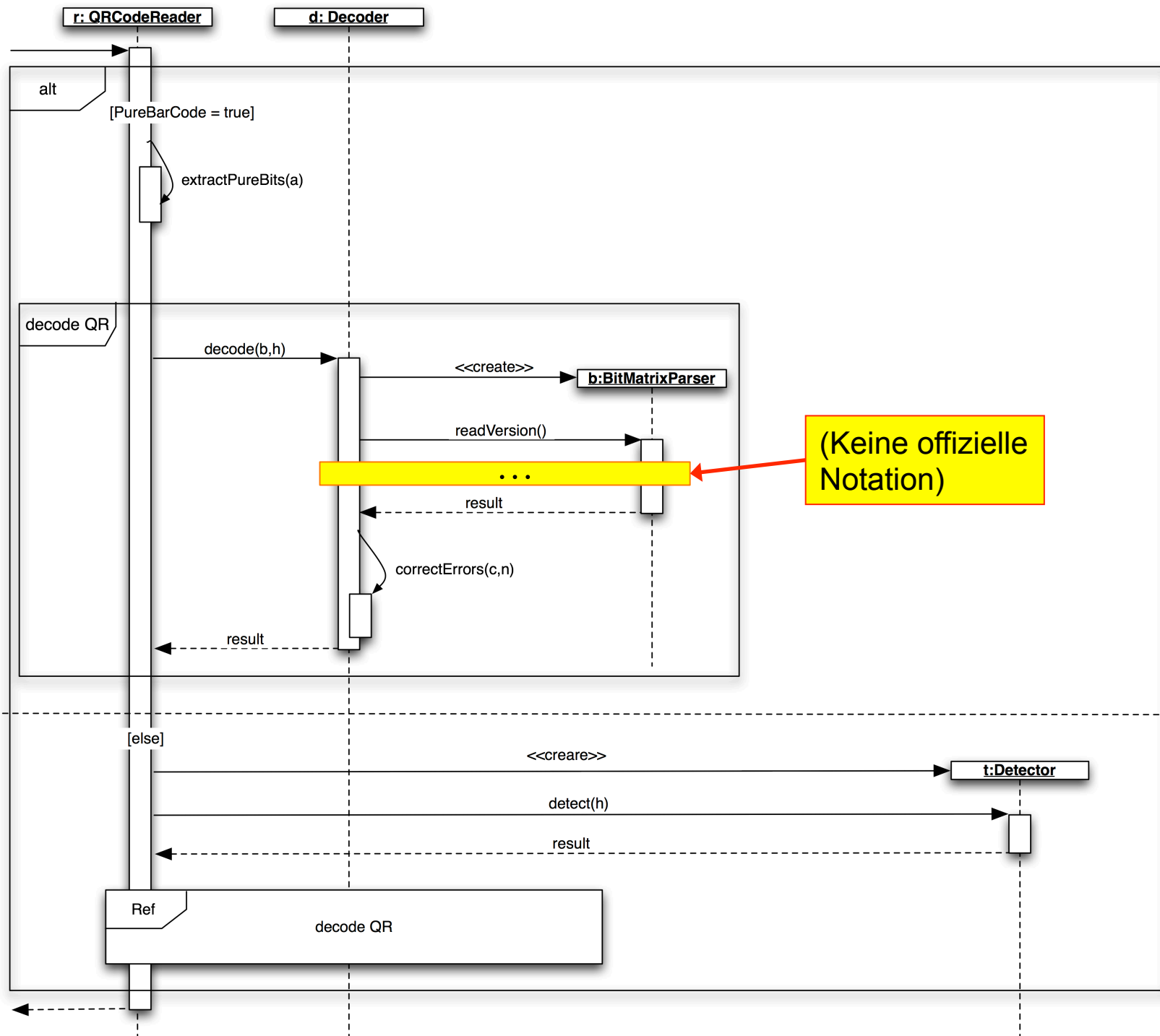
- Aktivierungsbalken (bei laufenden Methode)
- **Konstruktoraufruf**
- Alternativen



Aufgabe 3.3 – UML Sequenzdiagramm

- Aktivierungsbalken (bei laufenden Methode)
- Konstruktoraufruf
- **Alternativen**





Aufgabe 4.1 – Dokumentation

- Prüfziffer-Berechnung
 - Von rechts beginnen, jede zweite Nummer addieren
 - Ergebnis * 3
 - Von rechts, ab dritter Nummer beginnen, jede zweite Nummer addieren
 - Ergebnis modulo 10 rechnen, und dies von 10 subtrahieren

Aufgabe 4.1 – 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 4.1 – Dokumentation

- JavaDoc: Blackbox → **was** macht die Klasse / die Methode (und nicht **wie**)
- JavaDoc Kopfkomentare 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 4.1 – 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 4.1 – 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 4.1 – 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)
- Manchmal ist @pre und @post auch leer:
@pre -, @post -
- Z.B. für *checkStandardUPCEANChecksum(String s)*:

Aufgabe 4.1 – 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)

```
/**
```

```
* Checks whether the given barcode is a valid  
* standard UPC/EAN barcode by computing the checksum.  
* @pre s contains only numeric characters  
* @post s@pre = s@post  
* @param s the barcode to check  
* @return true if the barcode is valid, false otherwise  
* @throws FormatException if the given string s contains non-numeric characters  
*/
```

```
private static boolean checkStandardUPCEANChecksum(String s) throws FormatException {
```

Aufgabe 4.1 – 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 4.1 – Dokumentation

- Oft vergessen: Auch Kommentar innerhalb der Methode ist Methodenkomentar!
- Aber: wenn der Kommentar bloss das sagt, was schon offensichtlich ist, dann besser weglassen

```
//checks if the length of the string is bigger than zero  
If( string.length() > 0 )
```

Aufgabe 4.2 – Erweitern des Codes

- `rotateImage` if (`image.width < image.height`)
 - Ungültige Annahme, dass alle Bilder dieses Kriterium erfüllen

Systematisches Programmieren

- Kommentare zum Code
- Einheitliche / “schöne” Formatierung des Codes
- Sinnvolle Namen für Variablen, Methoden etc.
- Verständlichkeit (in der Korrektur etwas unpassend “Simplicity”)

Aufgabe 4.2 – Erweitern des Codes

Mögliche Lösung:

```
public Result decode(BinaryBitmap image, Hashtable hints) throws NotFoundException {
    try {
        return doDecode(image, hints);
    } catch( NotFoundException e ) {
        return doDecode(image.rotateCounterClockwise(), hints);
    }
}
```

(Codes auf dem Kopf werden bereits richtig erkannt)

```

public void visitListPre(List list)
{
    switch(currentListKind = list.getListKind())
    {
        case UNORDERED:
            writer.print("<ul>");
            break;
        case ORDERED:
            writer.print("<ol>");
            break;
        case DEFINITIONS:
            writer.print("<dl>");
            break;
    }
}

public void visitSectionPre(Section section) {
    switch(section.getDepth())
    {
        case 0: writer.println("<h2>" +section.getTitle()+"</h2>"); break;
        case 1: writer.println("<h3>" +section.getTitle()+"</h3>"); break;
        case 2: writer.println("<h4>" +section.getTitle()+"</h4>"); break;
        default: writer.println("<h4>" +section.getTitle()+"</h4>");
    }
}

```

(Separate Zeile
für Zuweisung)

Einrückungen

Kommentare
fehlen

Klammer-
positionen

Neue Zeile?

Allgemeines

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