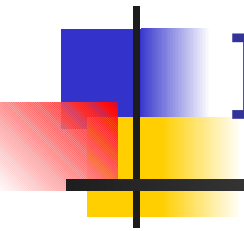
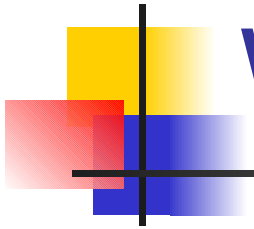


Implementierungs-Review II

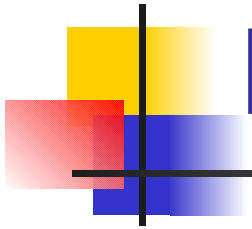
Ein Vortrag von

Evgueni Zinoviev
Manuel Haim
Marc Koch



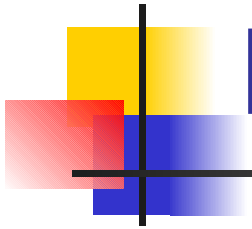
Was bislang geschah …

- In unserer letzten Präsentation
  - stellten wir alle unsere Klassen
  - und deren Methoden vor
  - Außerdem zeigten wir, wie man eine Bibel in eine Bohne bekommt



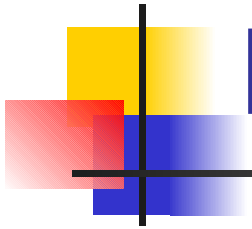
Kurze Erinnerung (1)

- de.fb12.apra05.al.*
  - Comment
  - Constants
  - File
  - LitEntry
  - LitList



Kurze Erinnerung (2)

- de.fb12.apra05.beans.*
  - DataBean (+ Unterklassen)
  - CommandBean (+ Unterklassen)
  - ResultBean



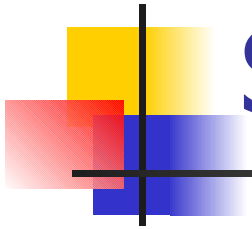
Kurze Erinnerung (3)

- Beans werden mit Daten gefüllt

```
LitEntryBean leb = new LitEntryBean();
leb.setAuthor("Mose");
```

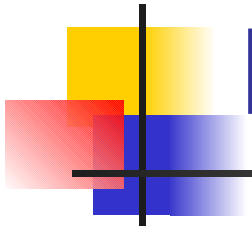
- und dann in den al-Klassen benutzt

```
public static ResultBean save (CommandBean cb){
 LitEntryBean db1 = (LitEntryBean)cb.getDataBean1();
 ...
 ResultBean rb = new ResultBean();
 ...
}
```



Seitdem hat sich was getan

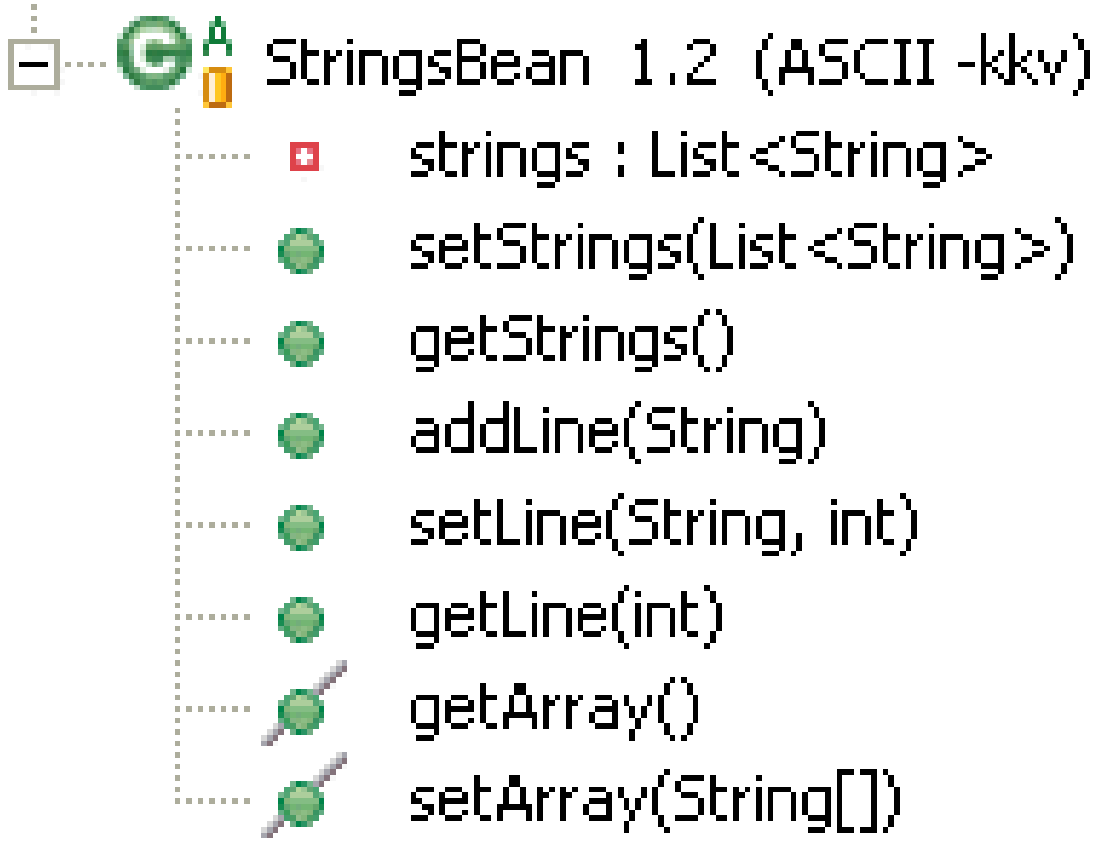
- Wir haben uns neue Klassen ausgedacht
  - apra05.al.BibTexEcxeption
  - apra05.al.BibTexHelper
  - apra05.beans.StringsBean
- Quelltext und Exception-Handling wurde hinzugefügt und verändert
- Arrays wurden zu Listen

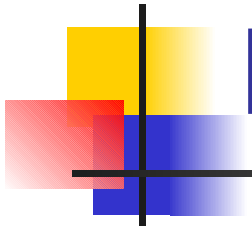


Die neuen Klassen

- BibTexException
  - Stellt eine Exception für BibTex bereit
- StringsBean
  - Fungiert als abstrakte Oberklasse für
    - BibTexBean
    - TextBean
    - XmlBean
    - SearchBean

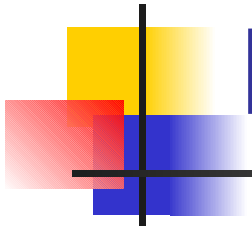
Die neuen Klassen: StringsBean





Die neuen Klassen

- BibTexHelper
  - Die ehemalige Klasse Constants
  - Erweitert um folgende Methoden
    - getBibTexFromString
    - getValue
    - getNextPos
    - texFilter
    - isBibTexChar
    - bibTexOut
    - textMasc



Exception Handling (1)

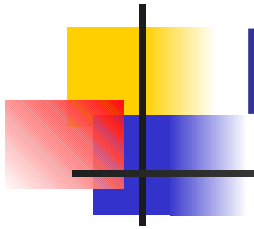
- Auftretende Exceptions werden gefangen
  - mittels try-catch
- in die Result Bean gepackt
  - momentan allerdings nur als String
- und an die Multiuser weitergereicht
  - Viel Spaß damit ;)

Exception Handling (2)

- Ein Beispiel anhand der Methode Comment.create

```
public static ResultBean create (CommandBean cb){
   Result Bean rb = new ResultBean();
   try {
   ...
   } catch (JDBCBeanException e) {
   rb.setException(„Kommentarerstellungs-Fehler");
   return rb;
}
```

Exception Handling (3)



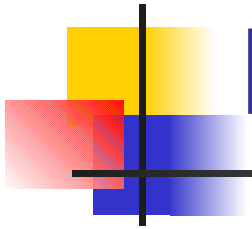
- Erklärung des Quellcodes
  - Erzeugung einer neuen ResultBean. Diese wird sowohl im Erfolgsfall als auch im Falle einer Exception zurückgegeben

```
Result Bean rb = new ResultBean();
```

  - Ausführung des try-Blocks

```
try {...}
```

Exception Handling (4)

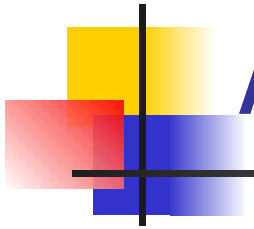


- Bei Auftreten einer Exception wird diese mit catch-Anweisung aufgefangen

```
catch (JDBCBeanException e) {
```

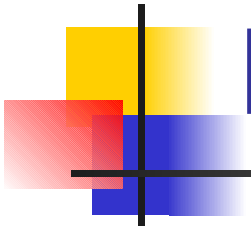
- Exception wird in die ResultBean gepackt und weitergereicht

```
rb.setException("Fehler");
return rb;
}
```



Allgemeines zur Verwendung

- Erzeugen, Füllen, Auslesen von
  - Listen
  - ArrayBean<E>
  - CommandBean
  - ResultBean



Listen: Erzeugung

- java.util.List<E> ist nur ein Interface!
  - man muss sich für eine Klasse entscheiden (z.B. ArrayList<E>, LinkedList<E>)
- aber alle Listen haben gleiche Methoden
  - add(e) fügt ein Element hinzu
  - addAll(list) fügt eine ganze Liste hinzu

```
List<String> newList;
newList = new java.util.ArrayList<String>();
newList.add("Listeneintrag 1");
```

Listen: Auslesen

- entweder über einen Iterator
  - wird über Methode iterator() angefordert
- oder über get-Methode
  - get(index) liefert ein Element

```
Iterator<String> it = newList.iterator();
while(it.hasNext()) {
  System.out.println(it.next());
}
//oder so:
String s = newList.get(0);
```

Listen bei StringsBean

- Alle StringsBean-Unterklassen haben bereits eine leere Liste
  - kann mit addLine(string) gefüllt werden
  - oder mit setStrings(list) gesetzt werden

```
TextBean tb = new TextBean();
tb.addLine("Listeneintrag 1");
tb.addLine("Listeneintrag 1");
//oder:
tb.setStrings(newList);
```

ArrayBean<E>: Erzeugen

- Kapselt DataBean-Array als DataBean
- E muss eine DataBean-Klasse sein
- Füllen über setDataBeans(dataBeans[])

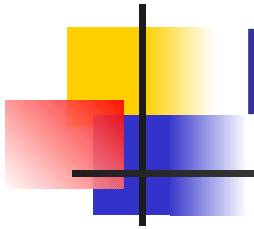
```
ArrayBean<LitListBean> ab = new ArrayBean();
LitListBean[] llbs = new LitListBean[] {
  new LitListBean(),
  new LitListBean()
};
ab.setDataBeans(llbs);
```

ArrayBean<E>: Auslesen

- Entweder über Iterator
  - dazu gibt's die Methode iterator()
- oder über Array getDataBeans()

```
Iterator<LitListBean> it = ab.iterator();
while(it.hasNext()) {
  LitListBean llb = it.next();
  System.out.println(llb.getListName());
}
//oder so:
LitListBean[] = ab.getDataBeans();
```

CommandBean und ResultBean



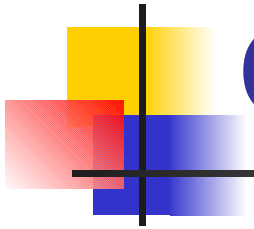
- CommandBean überträgt Befehle vom Webinterface an die Multiuser
  - enthält SessionBean und bis zu 2 DataBean's
  - Befehl ist durch DataBean-Kombination festgelegt
  - Multiuser prüfen Rechte des aufrufenden Users
  - und reichen die CommandBean bei bestehenden Rechten an die Anwendungslogik weiter
- ResultBean als „Antwort“
  - enthält die angeforderten Daten
  - oder eine Exception

CommandBean: Erzeugen

- Alle CommandBean-Unterklassen haben Konstruktoren für die Anwendungsfälle
  - z.B. CmdLoadBean(SessionBean, LitListBean)

```
SessionBean sb;
UserBean ub;
//...
sb.setUserBean(ub);
LitListBean llb = new LitListBean();
llb.setLitListKey("1457");
CmdLoadBean cb = new CmdLoadBean(sb, llb);
```

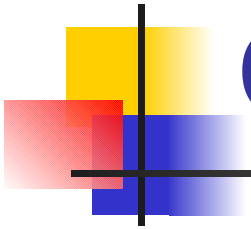
CommandBean: Aufrufen



- Eine CommandBean wird von WI an de.fb12.apra05.mu.BeanProcessor weitergereicht
  - Methode BeanProcessor.processBean(cb)
- MU „entschlüsselt“ die CommandBean und ruft passende AL-Methode auf
- Rückgabewert ist eine ResultBean
  - bei fehlenden Rechten von MU erzeugt
  - sonst von AL erzeugt

CommandBean: Aufrufen

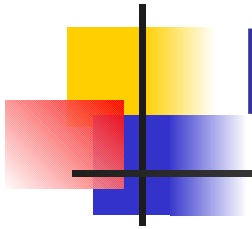
```
ResultBean rb;
rb = BeanProcessor.processBean(cb);
if(rb.getExeption().equals("")) {
   llb = (LitListBean) rb.getDataBean1();
   System.out.println("Name der LitList:");
   System.out.println(llb.getListName);
} else {
   System.out.println(rb.getExeption());
}
```



CommandBean: Auslesen

- Auslesen mittels getter-Methoden
  - getSessionBean()
  - getDataBean1() und getDataBean2()
- bei DataBean's Typecasting erforderlich
  - z.B. (LitListBean) cb.getDataBean1()
  - man muss aber vor dem Typecasting wissen, was für eine DataBean drin steht
    - Klasse abfragen: cb.getDataBean1().getClass()
    - das machen die Multiuser für uns :)

ResultBean: Erzeugen



- Einfach mittels setter-Methoden

```
ResultBean rb = new ResultBean();
LitListBean llb = new LitList();
rb.setDataBean1(llb);
```

Das war's von uns...

- Wir hoffen, es funktioniert alles :)