

# Inhalt

- 1 QTI 1.2(.1)
  - Aufbau eines Tests
  - Fragetypen
  - Aufbau eines Result Reports
- 2 Integration in OLAT - Der Editor
- 3 Integration in OLAT - Die Runtime
  - Darstellung (Rendering)
  - Laden eines Tests
  - Workflow
  - Erstellen des Result Reports

# Datenstrukturen

- ASI-Objekte (Assessment/Section/Item)
- beinhalten Elemente für Eigenschaften
- folgende Gruppierung ist festgelegt:
  - **Configuration:** legt Umgebung für korrekte Interpretation der im Objekt enthaltenen Daten fest
  - **Processing:** legt Präsetzung und Response Processing sowie Feedback fest
  - **Sequencing:** legt die Reihenfolge der Abarbeitung der enthaltenen ASI-Objekte fest

## ASI Object (Assessment / Section / Item)

### Configuration

<comments>  
<duration>  
<precondition>  
<postcondition>  
<metadata>  
<objectives>  
<control>

### Processing

<presentation>  
<processing>  
<feedback>

### Sequencing

<sub-objects>  
<ref-objects>  
<selection>  
<order>  
<reference>

## Item

- kleinste unabhängige Einheit
- repräsentiert genau eine Frage
- Test mit nur einem Item möglich

## Section

- kann weitere Sections und/oder Items enthalten
- wird genutzt um Gruppierungen festzulegen und
- definiert Bedingungen für die Reihenfolge der enthaltenen Objekte

## Item

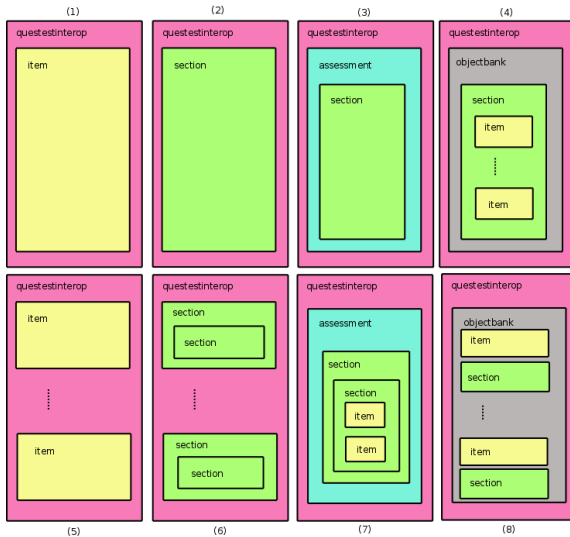
- kleinste unabhängige Einheit
- repräsentiert genau eine Frage
- Test mit nur einem Item möglich

## Section

- kann weitere Sections und/oder Items enthalten
- wird genutzt um Gruppierungen festzulegen und
- definiert Bedingungen für die Reihenfolge der enthaltenen Objekte

## Assessment

- stellt größtes Objekt dar, kein eigentlicher Bestandteil eines Tests
- dient als Überkontrollstruktur für Sections
- muss mindestens eine Section enthalten (keine Items erlaubt)
- es darf nur ein Assessment pro Test enthalten sein

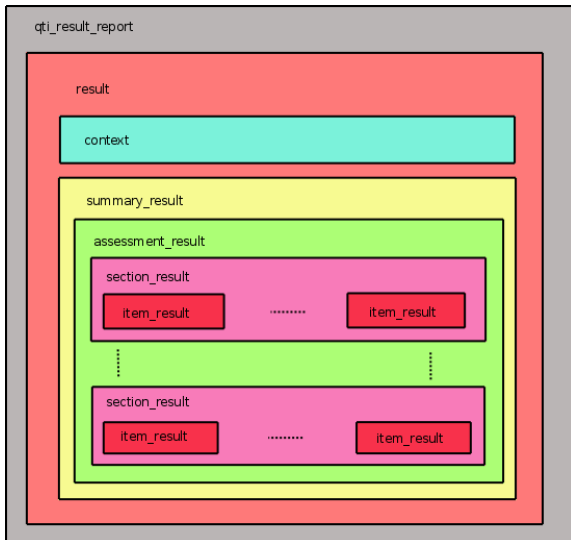


## Fragearten

- Wahr oder Falsch
- Multiple Choice mit einer richtigen Antwort (mit Bild, Ton, Slider)
- Multiple Response (MC mit mehreren richtigen Antworten)
- HotSpot-Rendering
- Ordnungsaufgaben
- Connect-the-Points
- Lückentext
- Freitext
- Drag-and-Drop

# Result Report

- enthält die Resultate zu einem Test
- besitzt folgenden Aufbau/Elemente:
  - **context**: kontextbezogene Daten zum Test (Name, Datumsangaben zu Start und Ende)
  - **summary\_result**: enthält alle Items des Tests
  - **assessment\_result**: ausführliche Informationen zum Gesamtverlauf des Tests
  - **section\_result**: ausführliche Informationen zu einer Sektion
  - **item\_result**: ausführliche Informationen zu einem Item



# OLAT-eigener Editor

- OLAT stellt eigenen Editor für Tests zur Verfügung
- unterstützt nur vier Frage-Typen: Single Choice, Multiple Choice, KPrim, Gap Text (Lückentext)
- keine Möglichkeit Subsections zu erstellen
- kann nur eigene Tests bearbeiten

The screenshot shows the OLAT editor interface for creating a new question. The main window is titled "KEE: QTI-Test" and has a tabbed interface with "Meta data" selected. The "Meta data" section contains the following fields:

- Title: \* New question
- Type: Single choice
- Description: (empty text area)
- Alignment of answers: Vertically (dropdown menu)
- Limit number of attempts?:  Yes  No

On the right side, there are three panels:

- Editor tools:** Preview, Close/save
- Add:** Section, Single choice, Multiple choice, Kprim, Gap text
- Change:** Delete, Move, Copy

On the left side, there are navigation buttons: "New section" and "New question".

# Runtime

- fest in OLAT integriert (kein Modul)
- Darstellung erfolgt komplett über Rendering-Workflow von OLAT
- kann fast alle Features von QTI 1.2.1 handhaben/darstellen
- erlaubt Wiederaufnahme eines Test nach Verbindungsverlust
- Aktualisierung der Ansicht nur nach Nutzerinteraktion

# Darstellung

- Aufbau der Darstellung durch ein VelocityContainer, genauer Template, festgelegt
- zusammengesetzt aus verschiedenen Komponenten

The screenshot displays a web-based test interface. At the top, it shows 'KEE: QTI-Test' and 'Actual score: 0 / 1' with a progress bar. Below this, the test content is presented in a structured way:

- KEE: QTI-Test**
- 1. Addition**
- 1.1. 2+2**

The main content area contains the following text and options:

**2+2**  
Wieviel ist 2+2?

2  
 3  
 1

At the bottom of the content area is a 'Save answer' button.

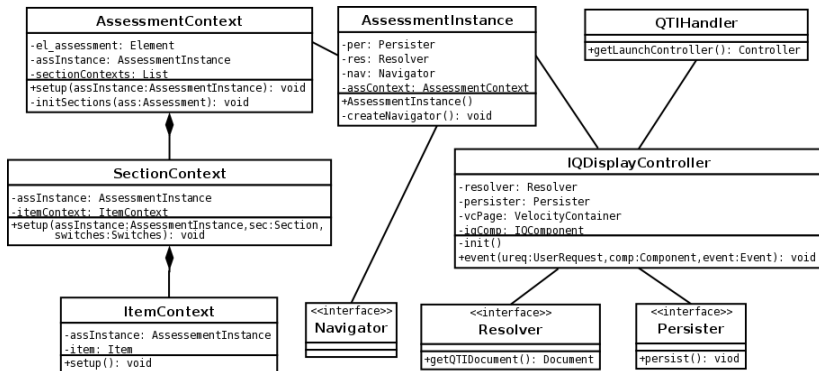
## Involvierte Klassen

- **IQDisplayController**: steuert gesamten Workflow
- **IQComponent**: “Behälter” für Test (AssessmentInstance), event-auslösende Komponente
- **AssessmentInstance**: beinhaltet Test (AssessmentContext), Persister, Resolver
- **AssessmentContext**: repräsentiert eigentlichen Test
- **Navigator**: steuert die Navigation, automatische Speicherung nach jedem Submit
- **ProgressBar**: Punkte- und Fragefortschrittsbalken
- **IQManager**: Erstellen der XML-Dokumente, Auswertung des Requests

## Laden eines Tests

- RepositoryEntry stellt im OLAT-System eine Lernressource dar (wird in Datenbank mit Hilfe von Hibernate gespeichert)
- über RepositoryEntry wird ermittelt, in welchem Ordner sich Test befindet
- Einlesen des Tests:
  - Einlesen der XML-Dateien mit Hilfe von DOM4J
  - AssessmentContext.setUp(AssessmentInstance ai): verarbeitet <assessment>-Element
  - für jede Section wird SectionContext erzeugt und jeweiliges <section>-Element übergeben, Speicherung in Collection  
*Code: new SectionContext().setUp(ai, sec\_elem, switches);*
  - jeder SectionContext verarbeitet wiederum das <section>-Element und erzeugt für jede enthaltene Frage ein ItemContext, werden in Collection gespeichert  
*Code: new ItemContext().setUp(ai, item\_elem, switches);*

- abschließend wird alles über den OLAT-Rendering-Prozess dargestellt



# Workflow

- sämtliche Steuerung erfolgt über event-Methode des IQDisplayController
- je nach Aktion werden unterschiedliche Funktionen abgearbeitet

## Beispiel: Absenden einer Frage

```
if (wfCommand.equals(" sitse" )) {  
    navig.submitItems(iqm.getItemsInput(ureq));  
    if (ai.isClosed()) {\  
        event(ureq, source, new Event(QTIConstants.QTI_WF_SUBMIT));  
        return;}}}
```

- nach jeder Aktion werden IQStatus und ProgressBars aktualisiert
- erneuter Rendering-Prozess

# Erzeugen eines Result Reports

- IQManager erzeugt DOM4J-Dokument anhand des AssessmentContext
  - ResultsBuilder wird erzeugt, welcher AssessmentContext auswertet und Document erstellt
  - Bewertung erfolgt anhand des ScoreModels, Runtime unterstützt lediglich zwei Arten (SumOfScores, NumberCorrect)
  - bei Verwendung anderer ScoreModels wird Fehler erzeugt
- FilePersister speichert XML-Datei in olatdata/resreporting/\$username/...