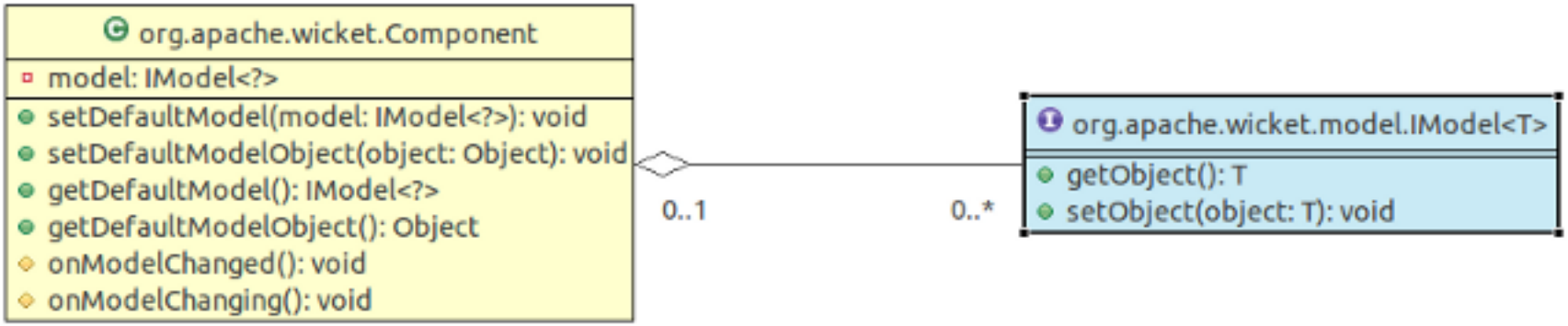
Die Bestandteile von Wicket

MODELS

Models

Grundlagen

- Kapselung von Datenobjekten (Strings, POJOs, …)
- **Data Binding**: UI-Komponenten ↔ fachliche Daten
- Höchstens ein Model pro Komponente



Models

Grundlagen

- Convenience: *Model.of(…)*
- Best Practices
  - Immer Models für Komponenten verwenden
  - Datenobjekte nur Auspacken, wenn benötigt

Models

anderScore trust in competence

GFU Cyrus AG

PropertyModel

- Kapselt einzelne Property eines Fachobjektes
- Vorteile gegenüber *Model.of(String)*
  - Immer aktueller Wert in UI
  - Aktualisierung der Property in Objekt

```
Person person = new Person();
//Personendaten laden...

Label label = new Label("name", new PropertyModel(person, "name"));
```

Models

anderScore trust in competence

GFU Cyrus AG

IModel mit Lambdas

- IModel ab Wicket 8 @FunctionalInterface
- Implementierung durch Lambda oder Methodenreferenz
- Typsicher, unterstützt Refactorings
- Read-only

```
Person person = new Person();
//Personendaten laden...

Label label1 = new Label("name1", person::name);
Label label2 = new Label("name2", () -> person.getName());
```

Models

anderScore trust in competence

GFU Cyrus AG

LambdaModel

- Neu in Wicket 8
- Alternative zu PropertyModel
- Typsicher, unterstützt Refactorings
- Höhere Performance

```
Person person = new Person();
//Personendaten laden...

IModel<String> nameModel = LambdaModel.of(person::getName, person::setName);
TextField<String> textField = new TextField<>("name", nameModel);
```

Models

Model Inheritance

- Interface *IComponentInheritedModel*
- Komponenten können Models an Nachfahren vererben
- Komponente hat kein Modell → Suche bei Vorfahren

Models

CompoundPropertyModel

- Implementiert *IComponentInheritedModel*
- Bindung üblicherweise an Container
- Funktion
  - Fungiert als „PropertyModelFactory“
  - ID der Komponenten-Nachfahren bestimmt Name der Property

Models

PropertyModel und…

```
Person person = new Person("John", "Smith");

add( new Label("name", new PropertyModel(person, "name")));
add( new Label("surname", new PropertyModel(person, "surname")));
add( new Label("address", new PropertyModel(person, "address")));
add( new Label("email", new PropertyModel(person, "email")));
add( new Label("spouseName", new PropertyModel(person, "spouse.name")));
```

Models

… CompoundPropertyModel

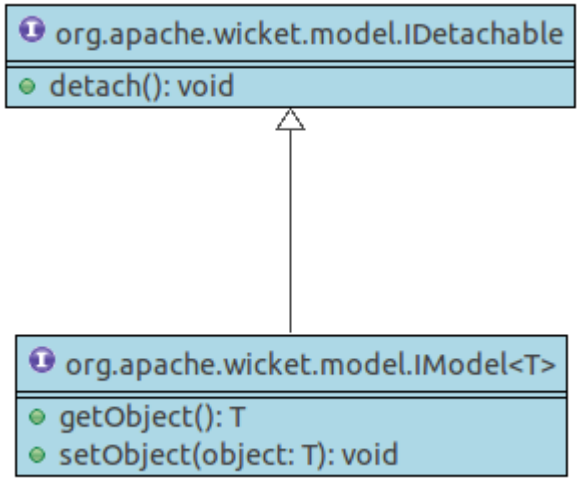
```
Person person = new Person("John", "Smith");
setDefaultModel(new CompoundPropertyModel(person));

add(new Label("name"));
add(new Label("surname"));
add(new Label("address"));
add(new Label("email"));
add(new Label("spouse.name"));
```

Models

Detachable Models

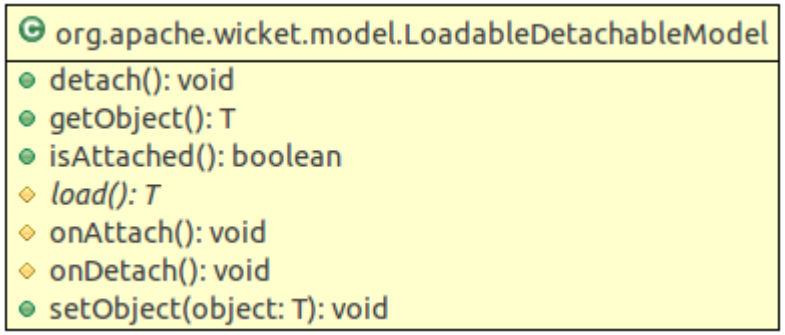
- Serialisierung von Models zusammen mit Pages
- Probleme bei unbekannten Datenobjekten
  - Größe
  - Serialisierbarkeit
- Lösung: Datenobjekte vor Serialisierung aushängen (z.B. in DB)



Models

LoadableDetachableModel

- Wiederherstellung des Datenobjektes (z.B. aus DB)
- Exemplarischer Workflow in Wicket
  - Datenobjekt aushängen, ID speichern
  - Page und Model serialisieren
  - Page und Model deserialisieren
  - Datenobjekt mit ID neu laden



Models

Model Chaining

- Interface *IChainingModel*
- Model als Datenobjekt eines anderen Models
- Beliebig tiefe Verkettung
- Anwendungsbeispiel: Abhängige Formularfelder