

- **Tag 1 – Einführung**

- Installation
- Erste Anwendung
- Architektur

- **Tag 2 – Entwicklung**

- Models
- Darstellung
- Formulare
- Ajax



- **Tag 3 – Fortgeschrittene Themen**

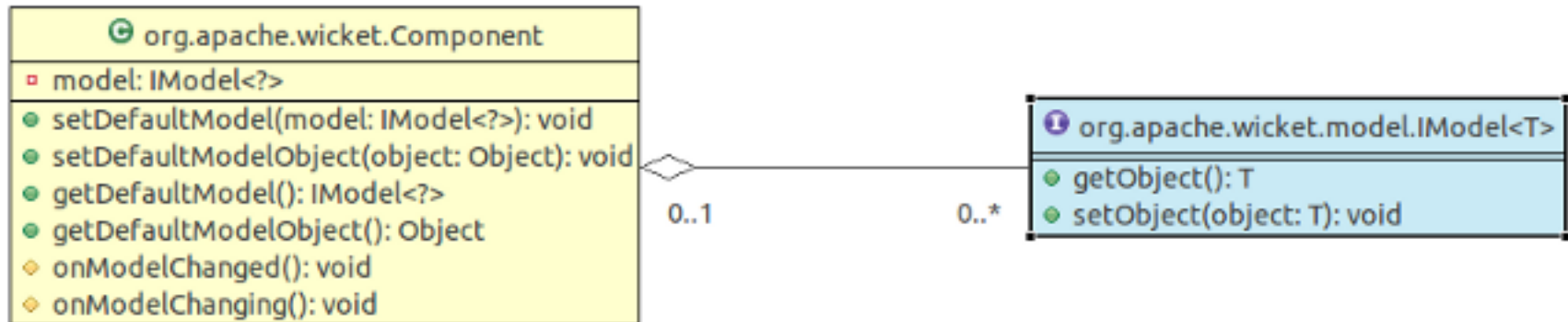
- Tests
- Security & Deployment
- Lokalisierung & Internationalisierung
- Performance
- Best Practices

## Die Bestandteile von Wicket

# MODELS

## Grundlagen

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



## Grundlagen

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

## 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"));
```

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

## 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);
```

## Model Inheritance

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



## CompoundPropertyModel

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

## PropertyModel und...

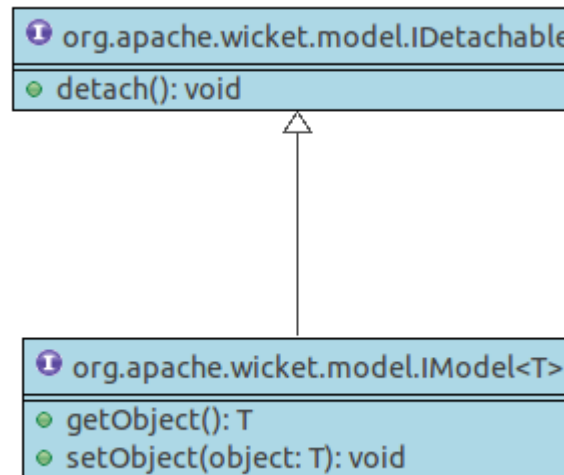
```
Person person = new Person("John", "Smith");  
  
add( new Label("name", new PropertyModel(person, "name")));  
add( new Label("surname", new PropertyModel(person, "surname")));  
add( new Label("address", new PropertyModel(person, "address")));  
add( new Label("email", new PropertyModel(person,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

## ... 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"));
```

## Detachable Models

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



## 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

org.apache.wicket.model.LoadableDetachableModel
detach(): void
getObject(): T
isAttached(): boolean
load(): T
onAttach(): void
onDetach(): void
setObject(object: T): void

## Model Chaining

- Interface *IChainingModel*
- Model als Datenobjekt eines anderen Models
- Beliebige tiefe Verkettung
- Anwendungen
  - Decorator
  - Abhängige Formularfelder

List of persons

Name:

Surname:

Address:

Email: