

# *eclipse* FORUM EUROPE 2008

21. bis 25. April 2008, Wiesbaden

Heiko Barth und Thorsten Schenkel, compeople AG

## Databinding in Eclipse



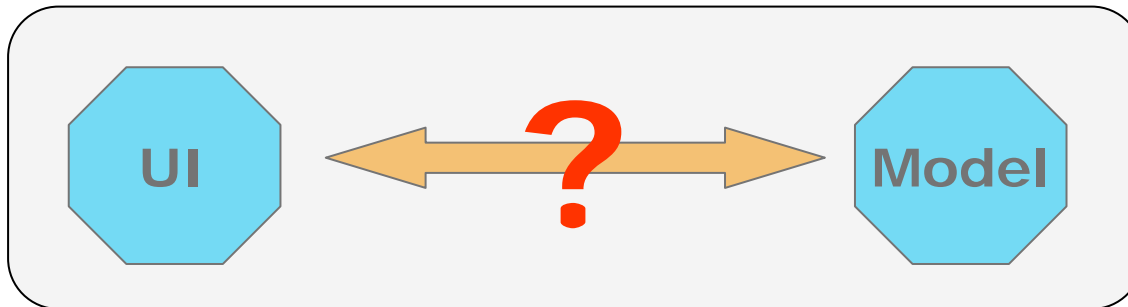
*Your Key to Success!*

# Agenda

- Vorstellung der grundlegenden Konzepte von DataBindings
- Ansatz von Eclipse DataBinding
- Die Kernkomponenten von Eclipse DataBinding
- Detaillierte Darstellung und Beispiele
- Abgeleitete Werte (Computed Values)
- ListView
- Pro und Contra Eclipse DataBinding

# Grundlegende Konzepte

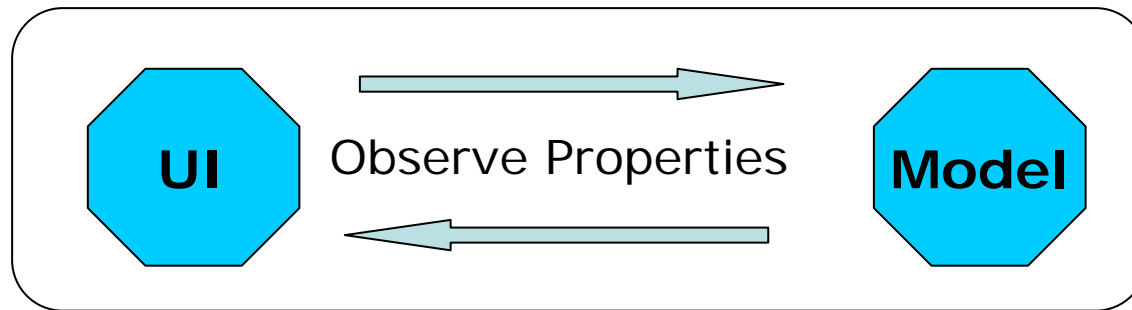
- Problem: Synchronisation von UI und Model



- Wie kann eine solche Verbindung umgesetzt werden?
- Die Lösung ist das Observer Pattern

# Grundlegende Konzepte

- Observer Pattern

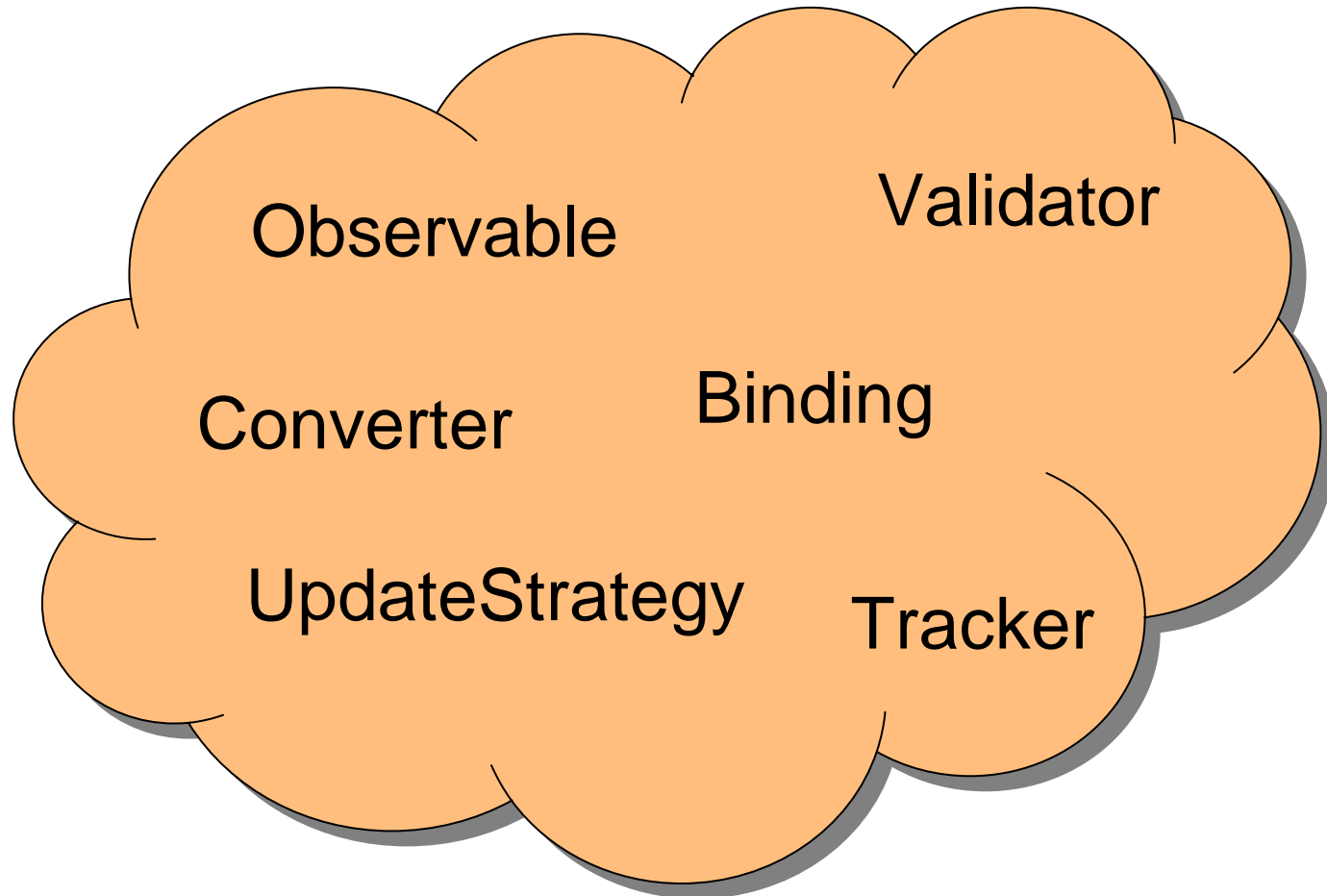


- Hauptprobleme bei manueller Vorgehensweise
  - Code-Redundanzen und Fehleranfälligkeit
  - Rekursionen
- Ansatzpunkt für Eclipse DataBinding

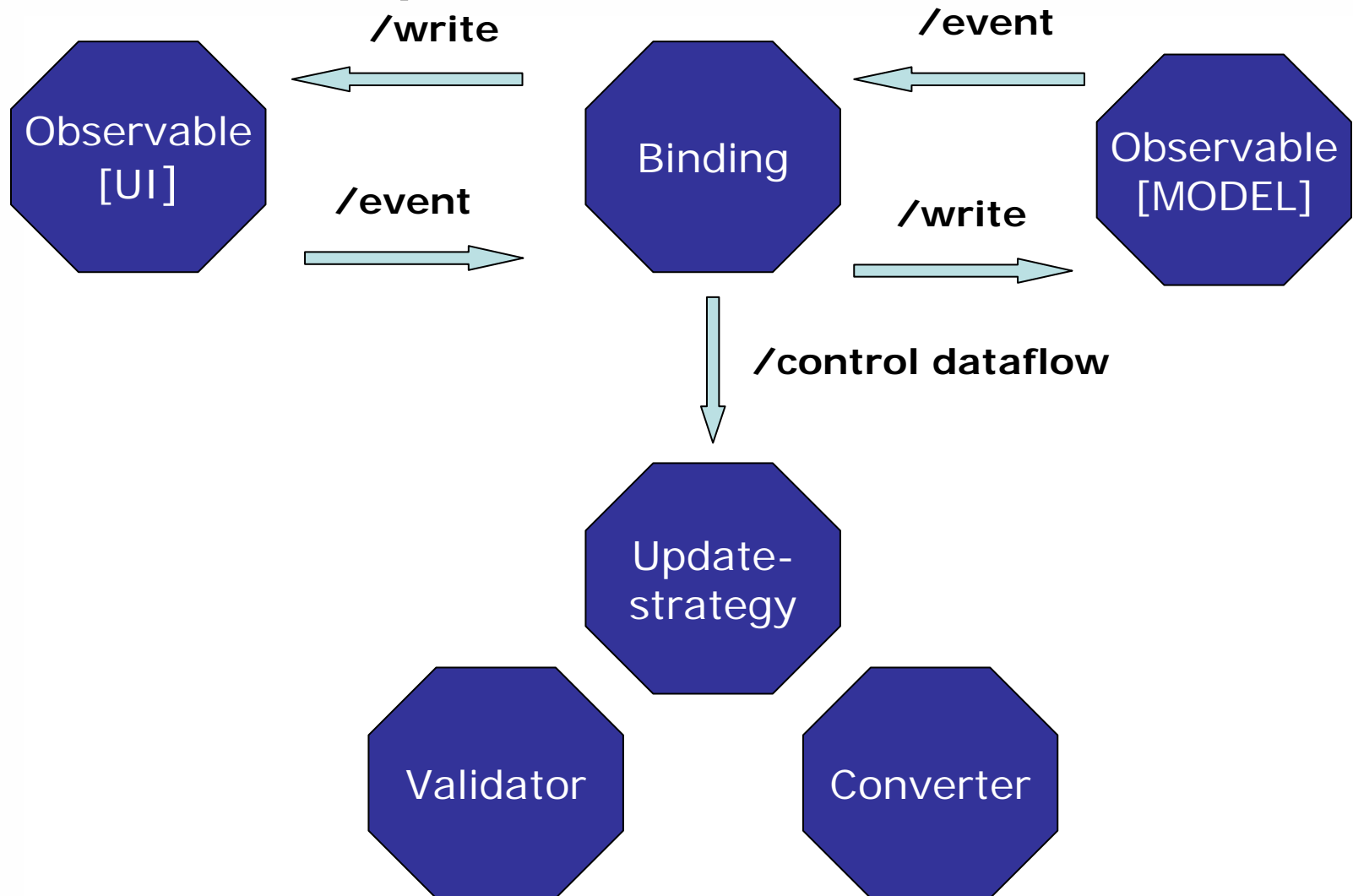
# Ansatz von Eclipse DataBinding

- Unterstützung für das Observieren und Binden von UI- und Modell-Komponenten
- Schutz vor Rekursionen
- Hooks für Daten-Validierung
- Hooks für Daten-Konvertierung
- Unterstützung im Bereich Master-Detail
- Interne Nutzung des Observer Patterns

# Kernkomponenten



# Kernkomponenten



# Observables

- **Beobachtung von Komponenten**
  - Model: Bean-Value (get-/setValue), List, Set, Map
  - UI: SWT Widgets (z.B. Text, List)
- **Factories für UI- und Modellobservation**
  - Model: BeansObservables (z.B. observeValue)
  - UI: SWTObservables (z.B. observeText)
- **Publikation von Änderungen durch Events**
- **Implementierung des ValuePattern**

## Observables

### BeansObservables

- Factory erzeugt Observables für Objekte, die der JavaBean Spezifikation entsprechen

	BeansObservables
	listFactory(Realm, String, Class)
	observeDetailList(Realm, IObservableValue, String, Class)
	observeDetailSet(Realm, IObservableValue, String, Class)
	observeDetailValue(Realm, IObservableValue, String, Class)
	observeList(Realm, Object, String)
	observeList(Realm, Object, String, Class)
	observeMap(IObservableSet, Class, String)
	observeMaps(IObservableSet, Class, String[])
	observeSet(Realm, Object, String)
	observeSet(Realm, Object, String, Class)
	observeValue(Object, String)
	observeValue(Realm, Object, String)
	setFactory(Realm, String)
	setFactory(Realm, String, Class)
	valueFactory(Realm, String)
	BeansObservables()

# Observables

## PropertyChangeSupport

- JavaBean unterstützt  
PropertyChangeSupport
  - addPropertyChangeListener
  - removePropertyChangeListener
  - Nach dem Setzen eines Wertes wird ein  
PropertyChangeEvent gefeuert
- BeansObservables können Änderungen im  
Model sofort propagieren.

## Observables

### Codebeispiel für BeansObservables


































```
model = new Person();  
IObservableValue firstNameOV_model = BeansObservables.observeValue(model, "firstName");
```



## Observables

### SWTObservables

- Factory erzeugt Observables für SWT Widgets

	SWTObservables
	 getRealm(Display)
	 observeBackground(Control)
	 observeEditable(Control)
	 observeEnabled(Control)
	 observeFont(Control)
	 observeForeground(Control)
	 observeItems(Control)
	 observeMax(Control)
	 observeMin(Control)
	 observeSelection(Control)
	 observeSingleSelectionIndex(Control)
	 observeText(Control)
	 observeText(Control, int)
	 observeTooltipText(Control)
	 observeVisible(Control)
	 SWTObservables()

# Observables

## Codebeispiel für SWTObservables

```
Text firstNameText = new Text(shell, SWT.BORDER | SWT.SINGLE);  
IObservableValue firstNameOV_ui = SWTObservables.observeText(  
    firstNameText, SWT.Modify);
```

# Binding

- Bindeglied zwischen 2 Observables
- Observation der Observables
- Uni- oder Bidirektional
- Propagation von Wertänderungen an Partner-Observable
- Unterteilung einer Synchronisation in Phasen
- Nutzung und Verwaltung von UpdateStrategies

## Binding

### Codebeispiel für Binding

```
IObservableValue firstNameOV_ui = SWTObservables.observeText(  
    firstNameText, SWT.Modify);  
IObservableValue firstNameOV_model = BeansObservables.observeValue(  
    model, "firstName");  
  
DataBindingContext context = new DataBindingContext();  
context.bindValue(firstNameOV_ui, firstNameOV_model, null, null);
```

# Binding

- Live Demo in Eclipse
  - Person Binding



PersonBinding

Person

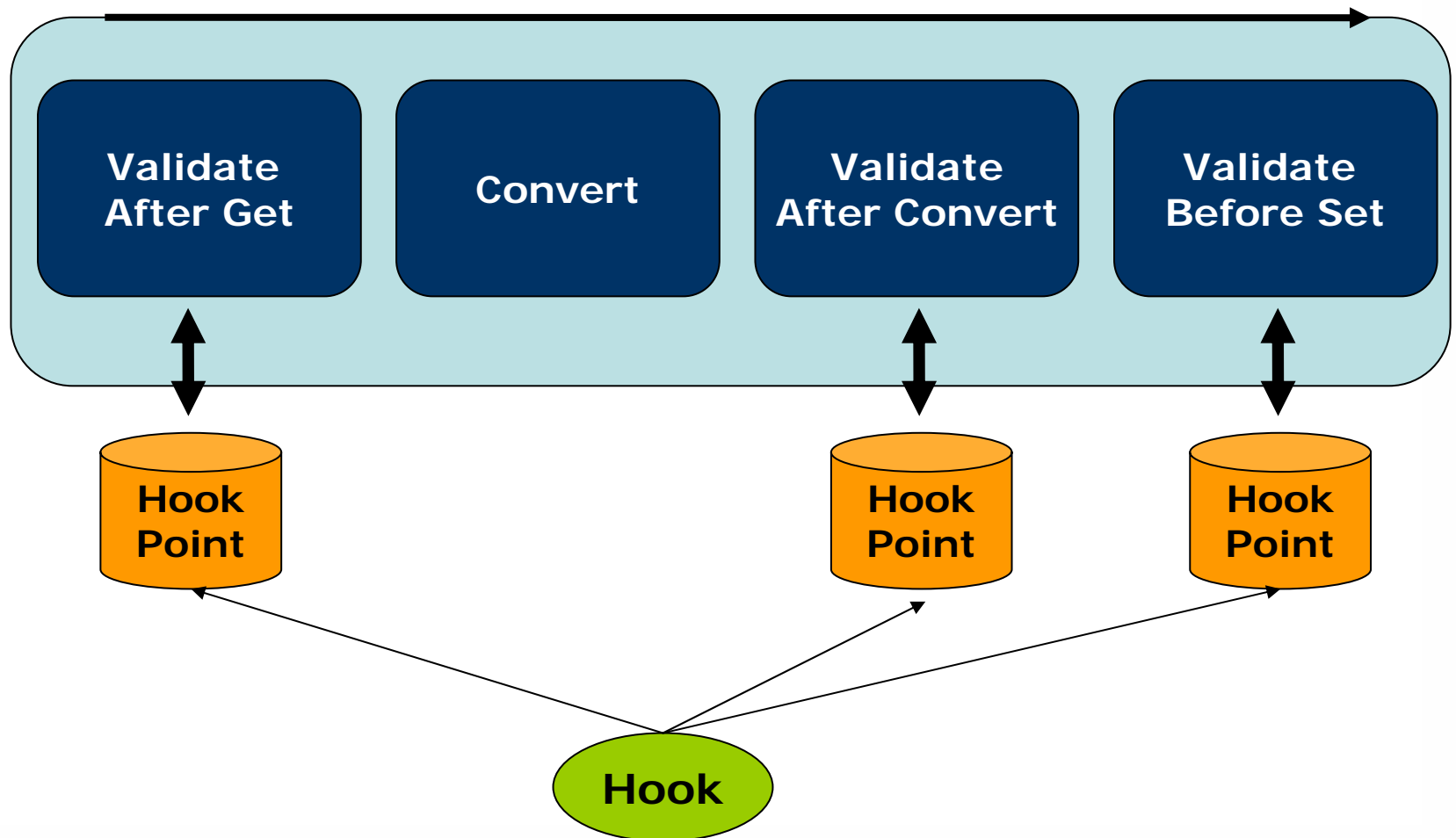
First Name

Last Name

Write HELLO To Model

# Binding

## Update-Phasen (ValueBinding)



# UpdateStrategy

- Container für Validatoren
  - afterGet
  - afterConvert
  - beforeSet
- Container für Converter
- Default-Converter für Basis-Typen
- Beim Binden kann für jede Datenflußrichtung eine UpdateStrategy gesetzt werden
  - targetToModel
  - modelToTarget

## Validator

- Validiert einen Wert
- Die Validierung liefert einen Status
  - *OK*
  - *CANCEL*
  - *INFO*
  - *WARNING*
  - *ERROR*
    - Datenfluß wird unterbrochen

```
public interface IValidator {  
  
    public IStatus validate(Object value);  
  
}
```

## Validator

### Codebeispiel für Validator (1. Teil)

- Validator-Klasse

```
public class NoUmlautValidator implements IValidator {  
  
    private final static String[] NOT_ALLOWED =  
        new String[] { "ä", "ö", "ü" };  
  
    public IStatus validate(Object value) {  
        String str = (String) value;  
        for (String umlaut : NOT_ALLOWED) {  
            if (str.contains(umlaut)) {  
                return ValidationStatus.error("Umlaut Error!");  
            }  
        }  
        return Status.OK_STATUS;  
    }  
}
```

## Validator

### Codebeispiel für Validator (2. Teil)

- Validator setzen (targetToModel)

```
// Validation
UpdateValueStrategy updateStrategy = new UpdateValueStrategy();
updateStrategy.setAfterGetValidator(new NoUmlautValidator());

// firstName
IObservableValue firstNameOV_ui = SWTObservables.observeText(
    firstNameText, SWT.Modify);
IObservableValue firstNameOV_model = BeansObservables.observeValue(
    model, "firstName");
context.bindValue(firstNameOV_ui, firstNameOV_model, updateStrategy,
    null);
```

# Validator

- Nach dem Holen der Werts aus dem Textfeld wird dieser validiert
- Im Fehlerfall (ERROR) wird der Datenfluss unterbrochen
  - Wert wird nicht in das Model geschrieben
  - Status wird im Binding-Kontext gesammelt

# Validator

- Status des Binding-Kontext observiert werden.
  - AggregateValidationStatus
    - Observiert Status der Liste von Bindings
    - Ist selbst eine ObservableValue
- AggregateValidationStatus kann z.B. mit JavaBean gebunden werden
  - Dieses Bean reagiert auf Änderungen des Status
    - Im Fehlerfall wird eine Meldung ausgegeben und der Wert zurückgesetzt

## Validator

### Codebeispiel für Validator (3. Teil)

- Status der Validierung auswerten

```
AggregateValidationStatus observableMaxStatus =
    new AggregateValidationStatus(
        context.getBindings(),
        AggregateValidationStatus.MAX_SEVERITY);

IObservableValue observableStatusHandler = BeansObservables
    .observeValue(new StatusHandler(), "status");

context.bindValue(observableMaxStatus, observableStatusHandler,
    null, null);
```

## Validator

### Codebeispiel für Validator (4. Teil)

- JavaBean, das Statusänderungen verarbeitet

```
private class StatusHandler {  
  
    // ...  
  
    public void setStatus(IStatus status) {  
        this.status = status;  
        if (!getStatus().isOK()) {  
            showErrorBox();  
            context.updateTargets();  
        }  
    }  
}  
  
// ...
```

# Validator

- Live Demo in Eclipse
  - ValidationBindingExample

Validation Sample

Person (ä, ö or ü are not allowed)	First Name	<input type="text"/>
	Last Name	<input type="text"/>

## Converter

- Konvertiert einen Wert
- Es existieren bereits ein Reihe von Konvertern
  - StringToShortConverter
  - StringToDateConverter
  - IntegerToStringConverter
  - Etc.
  - Siehe: `org.eclipse.core.internal.databinding.conversion`

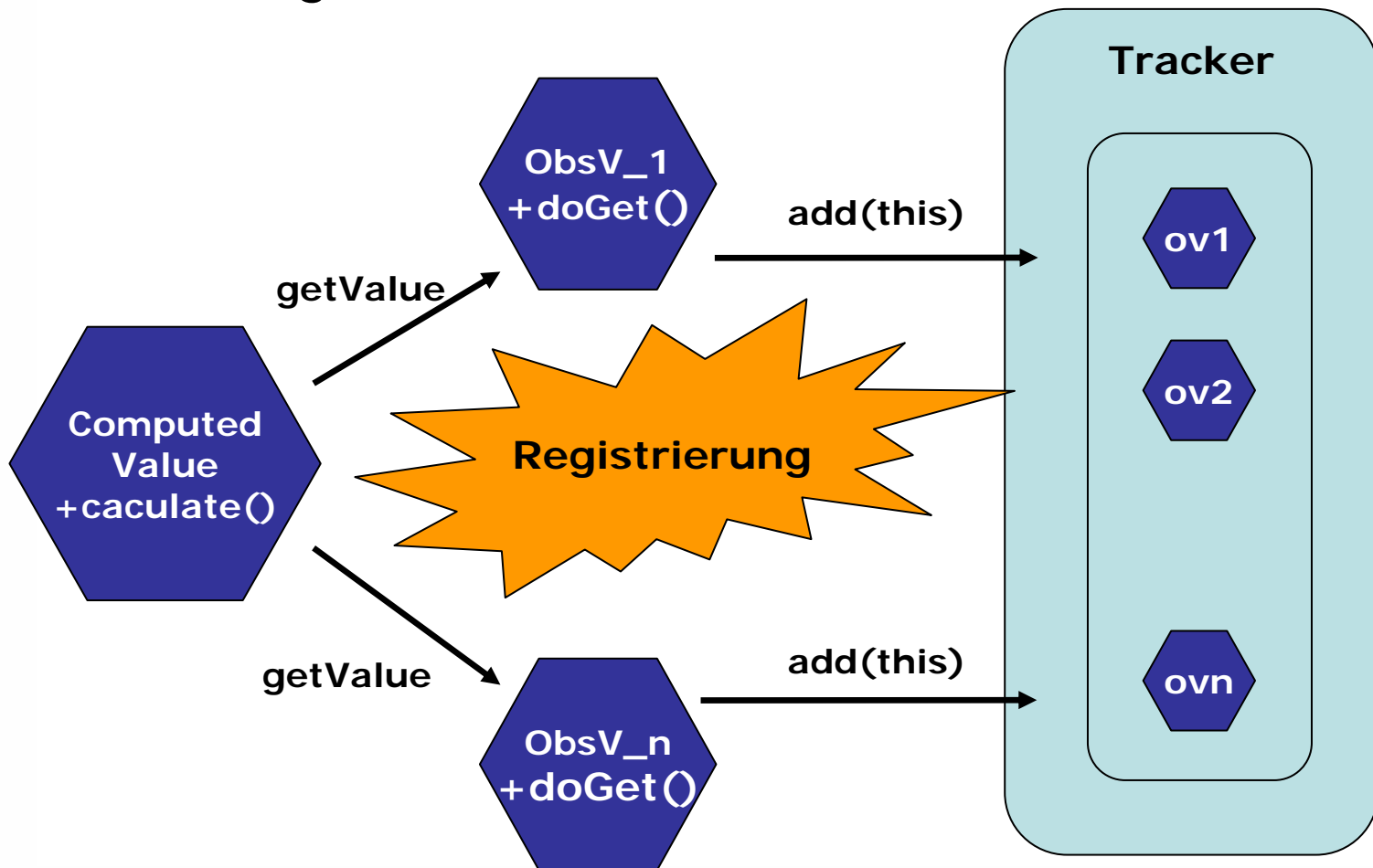
```
public interface IConverter {  
    public Object convert(Object fromObject);  
    public Object getFromType();  
    public Object getToType();  
}
```

# Computed Value

- Unterstützung für abgeleitete Werte
- Dynamische Ermittlung von Abhängigkeiten
- Technische Umsetzung
  - Tracking-Mechanismus

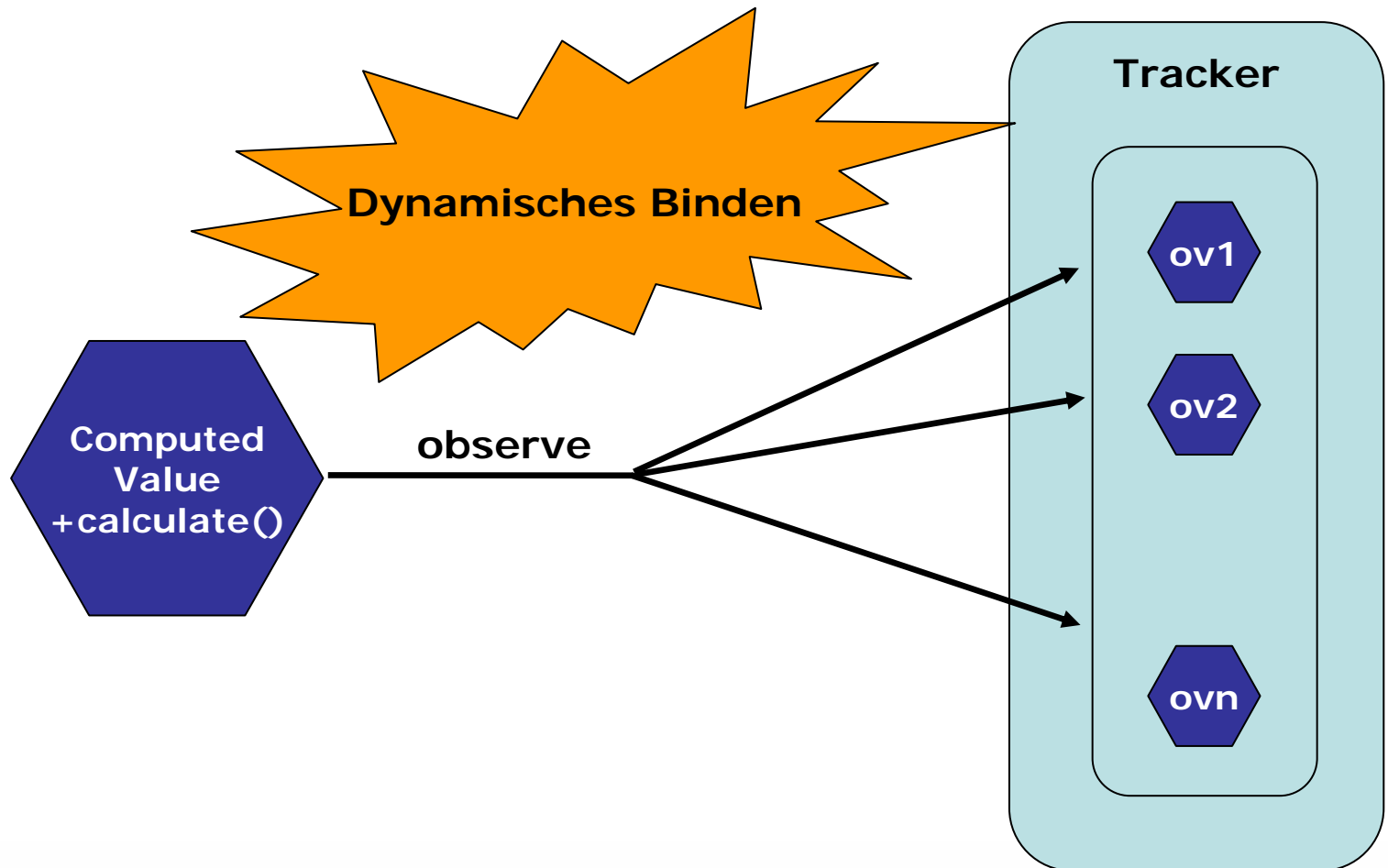
# Computed Value

## Tracking Mechanismus (Teil 1)



# Computed Value

## Tracking Mechanismus (Teil 2)



## Computed Value

### Codebeispiel

```
// computed value
IObservableValue computed_ui = SWTObservables.observeText(
    computedNameText, SWT.FocusOut);
IObservableValue computed_model = new FullNameObservable();
context.bindValue(computed_ui, computed_model, null, null);

private class FullNameObservable extends ComputedValue {
    @Override
    protected Object calculate() {
        String firstName = (String) firstNameOV_model.getValue();
        String lastName = (String) lastNameOV_model.getValue();
        return lastName + ", " + firstName;
    }
}
```

## List

- Live Demo in Eclipse
  - ComputedValueDemo

Observable Person	First Name	<input type="text" value="Paul"/>
	Last Name	<input type="text" value="Mayer"/>
	Full name	<input type="text" value="Mayer, Paul"/>

# List

- Binden von ListViewer (JFace)
  - Listeninhalt
    - ContentProvider:  
ObservableListContentProvider
    - Input: WritableList
  - Selektion
    - ViewersObservables
      - Für JFace Viewer, die ISelectionProvider implementieren

## List

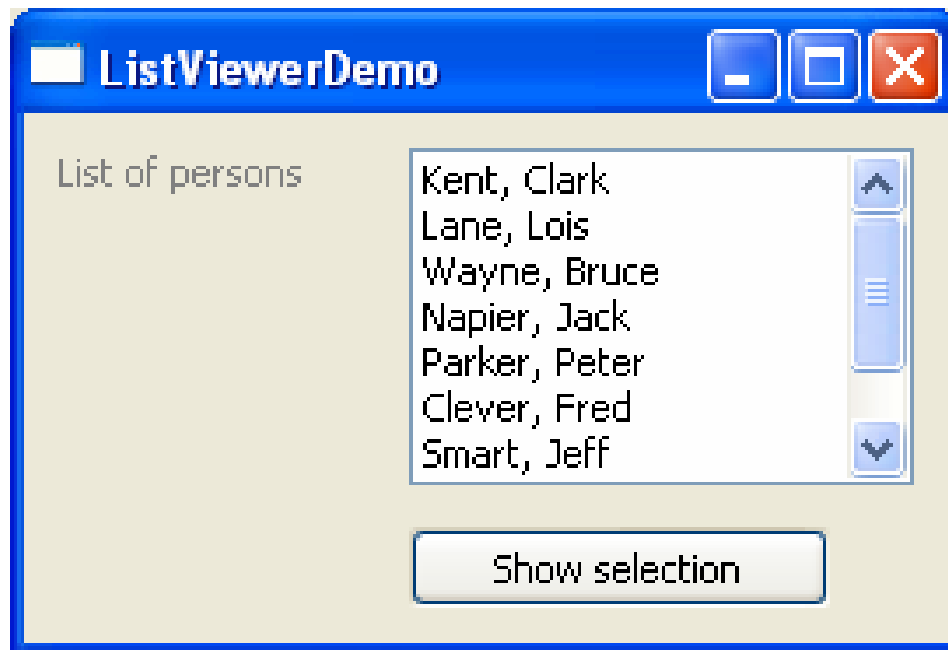
### Codebeispiel für Listen

```
// content
listViewer.setContentProvider(new ObservableListContentProvider());
IObservableList items_model =
    new WritableList(persons, Person.class);
listViewer.setInput(items_model);

// selection
IObservableValue selection_ui = ViewersObservables
    .observeSingleSelection(listViewer);
IObservableValue selection_model =
    BeansObservables.observeValue(model, "selection");
context.bindValue(selection_ui, selection_model,
    null, null);
```

# List

- Live Demo in Eclipse
  - ListViewerDemo



## Vorteile

- Minimierter Aufwand für die Synchronisation von GUI und Model
- Einfache Integration von Validatoren und Konvertern
- Erhöhte Wartbarkeit im Vergleich zu manueller Vorgehensweise

# Nachteile

- Zusätzliche Schicht
  - Erhöhter Debug-Aufwand
- Introspection und Reflection
  - Performance Einbußen
  - Refactoring Einschränkungen

## Resumé

- Einfaches und einheitliches Databinding-Konzept
- Weite Verbreitung in der Eclipse-Community
- UI-Toolkit unabhängig (SWT, Swing, GWT, ... )
- Seit eclipse 3.3 stabiles API

# Referenzen

- Eclipsepedia - JFace Data Binding  
[http://wiki.eclipse.org/JFace\\_Data\\_Binding](http://wiki.eclipse.org/JFace_Data_Binding)
- Eclipse Data Binding für die Kommunikation zwischen Modell und GUI  
Artikel von Ludwig Mittermeier im Eclipse Magazin  
4/2007

Vielen Dank für Ihre Aufmerksamkeit

**Besuchen Sie uns in der  
compeople Lounge 46**



[heiko.barth@compeople.de](mailto:heiko.barth@compeople.de)  
[thorsten.schenkel@compeople.de](mailto:thorsten.schenkel@compeople.de)