

JavaBeans

Property-Editoren und Customizer

© J. Heinzlreiter
WS 2004/05

Property-Editoren

- Builder Tools beinhalten *Standard-Editoren* für
 - Zahlen (ganze Zahlen, Gleitkommazahlen),
 - Strings,
 - Auswahl von Fonts,
 - Auswahl von Farben.
- JavaBeans API erlaubt das Hinzufügen von *benutzerdefinierten Editoren*.
 - Editor erlaubt die Änderung der Property mittels eines grafischen Dialogs.
 - Property kann in Property-Panel grafisch dargestellt werden.

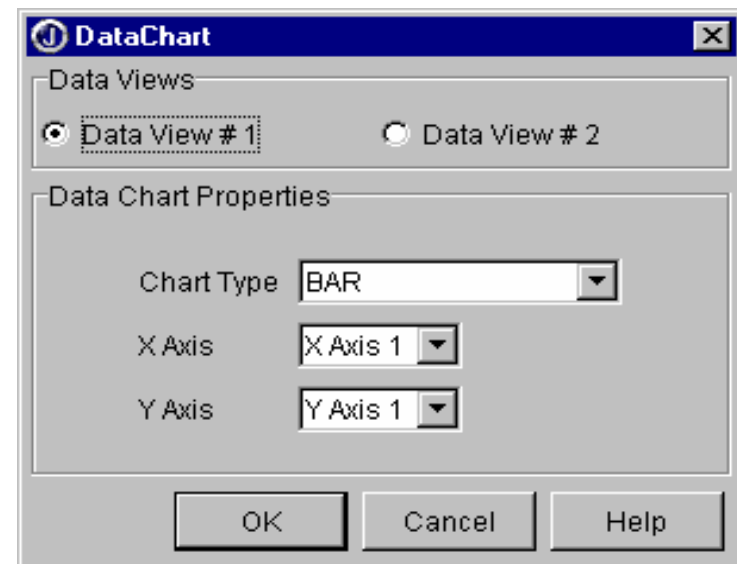
Arten von Property-Editoren

- Textbasiert
- Für Aufzählungen
- Mit benutzer-definiertem Dialog

legendVisible	False
maximumSize	32767, 32767
minimumSize	100, 100

legendAnchor	East
legendOrientation	Vertical
legendVisible	Horizontal
maximumSize	Vertical

chartAreaAppearance	Click to edit...
dataChart	{(data1 BAR)(data2 F ...
dataMisc	Click to edit...




Interface PropertyEditor (1)

```
public interface PropertyEditor {  
    public void setValue(Object value);  
    public Object getValue();  
    public boolean isPaintable();  
    public void paintValue(Graphics g,  
                           Rectangle box);  
    public String getJavaInitializationString();  
    public String getAsText();  
    public void setAsText(String s)  
        throws IllegalArgumentException;  
    public String[] getTags();  
    public boolean supportsCustomEditor();  
    public Component getCustomEditor();  
    public void addPropertyChangeListener(  
        PropertyChangeListener listener);  
    public void removePropertyChangeListener(  
        PropertyChangeListener listener);  
}
```

Interface PropertyEditor (2)

- **setValue/getValue**
 - Verwaltung des Werts der Property.
 - Property kann nur einen Wert verwalten.
- **getAsText/setAsText**
 - Repräsentierung der Property als String.
- **isPaintable/paintValue**
 - Graphische Darstellung der Property im Property-Panel.
- **getJavaInitializationString**
 - Java Codefragment zur Initialisierung der Property.
 - Beispiel: "new Color(255, 0, 0)"
 - bean.setColor(new Color(255, 0, 0));

precision	-1
scaleColor	 Orange
scaleExtent	1.0

Interface PropertyEditor (3)

- `getTags`
 - Mögliche Werte bei Aufzählungen.
- `supportsCustomEditor/getCustomEditor`
 - Liefert eine GUI-Komponente zum Editieren der Property.
- `add/removePropertyChangeListener`
 - Client (Builder Tool) wird von Änderungen des Property-Werts mit `propertyChange()` informiert.

Implementierung eines Property-Editors

- **PropertyEditor**
 - Interface `PropertyEditor` implementieren oder
 - von Klasse `PropertyEditorSupport` ableiten.

- **Registrierung in BeanInfo**

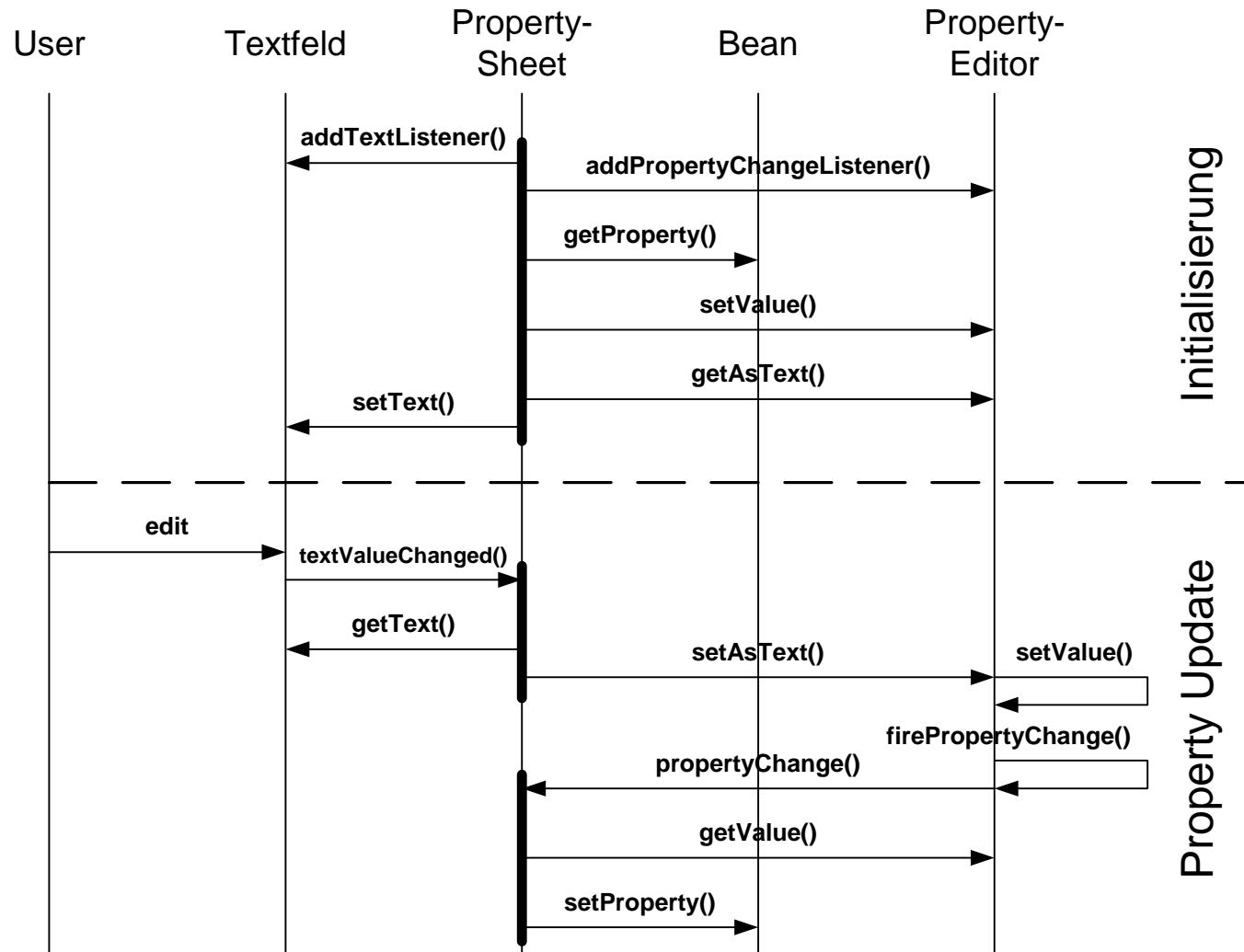
```
public class MyBeanBeanInfo extends SimpleBeanInfo {
    public PropertyDescriptor[] getPropertyDescriptors() {
        try {
            PropertyDescriptor prop1 =
                new PropertyDescriptor("myProp", MyBean.class);
            prop1.setPropertyEditorClass(MyPropEditor.class);
            return PropertyDescriptor[] { prop1 };
        }
        catch (IntrospectionException) { ... }
    }
}
```

Textbasierter Property-Editor

- Beispiel: Editieren eines Punkts: (x; y)

```
public class PointEditor extends PropertyEditorSupport {
    private final static MessageFormat pointFormat =
        new MessageFormat("({0, number}; {1, number})");
    public void setAsText(String s) {
        Object[] p = pointFormat.parse(s);
        setValue(new Point(((Number)p[0]).intValue(),
            (((Number)p[1]).intValue()));
    }
    public String getAsText() {
        Point p = (Point)getValue();
        return pointFormat.format(p.x(), p.y());
    }
    public String getJavaInitializationString() {
        Point p = (Point)getValue();
        return "Point(" + p.x() + ", " + p.y() + ")";
    }
}
```


Textbasierter Editor: Interaktion



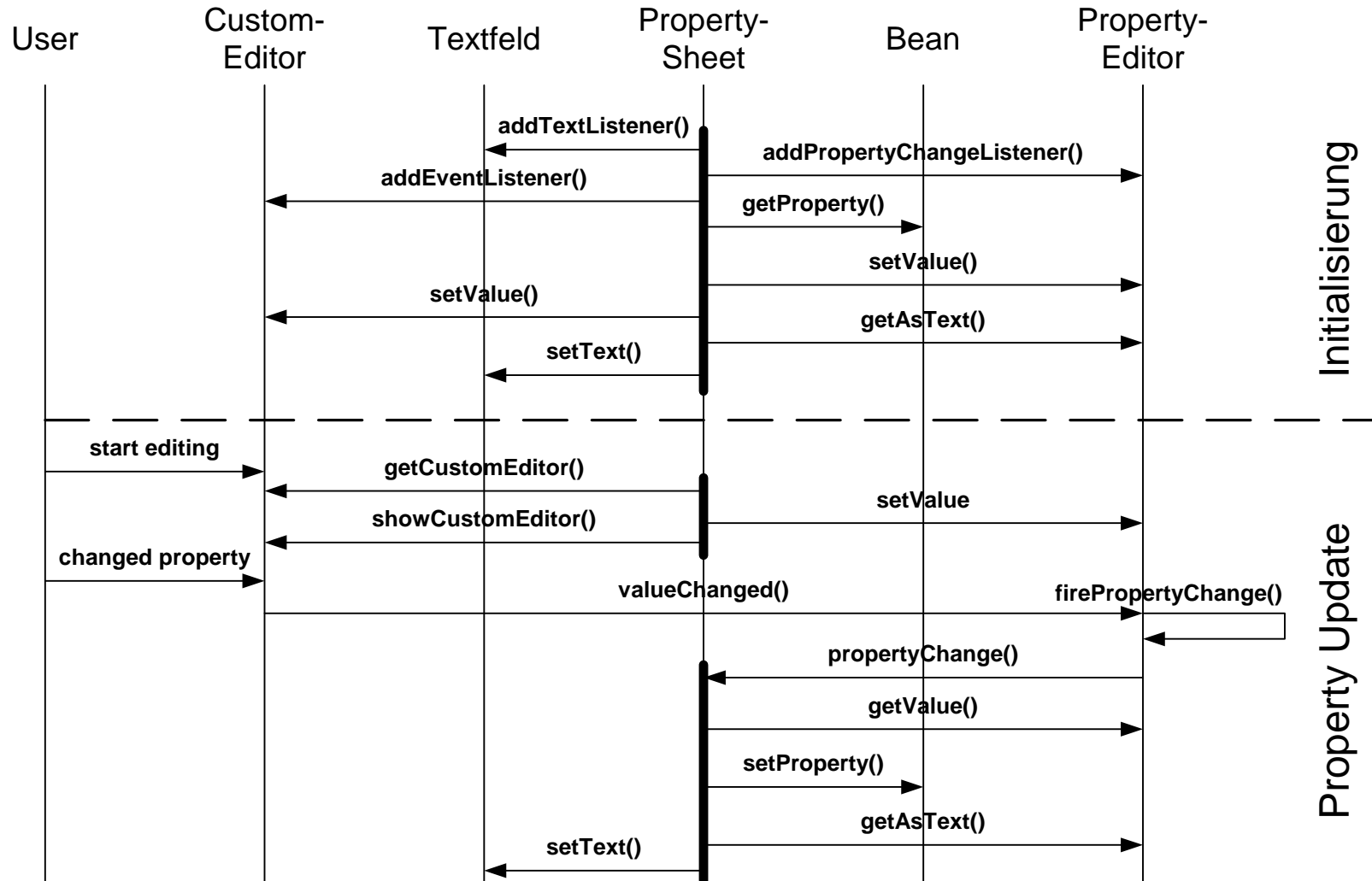
Property-Editor für Aufzählungen

```
public class AlignmentEditor extends PropertyEditorSupport {
    public String[] getTags() {
        return new String[] {"left", "right", "center"};
    }
    public void setAsText(String s) {
        if (s.equals("left"))
            setValue(new Integer(MyBean.LEFT));
        else ...
    }
    public String getAsText() {
        switch (((Integer)getValue()).intValue()) {
            case MyBean.LEFT: return "left";
            ...
        }
    }
    public String getJavaInitializationString() {
        switch (((Integer)getValue()).intValue()) {
            case MyBean.LEFT: return "MyBean.LEFT";
            ...
        }
    }
}
```

Editor mit benutzerdefiniertem Dialog

```
public class MyPropEditor extends PropertyEditorSupport {
    public String getAsText() { ... }
    public boolean supportsCustomEditor() { return true; }
    public Component getCustomEditor() {
        MyPropDialog dlg = new MyPropDialog(getValue());
        dlg.addMyListeners(new MyListeners() {
            public valueChanged(MyEvent e) {
                firePropertyChange();
            }
        });
        return dlg;
    }
    public boolean isPaintable() { return true; }
    public void paintValue(Graphics g, Rectangle clipRect) {}
    public String getJavaInitializationString() { ... }
}
```

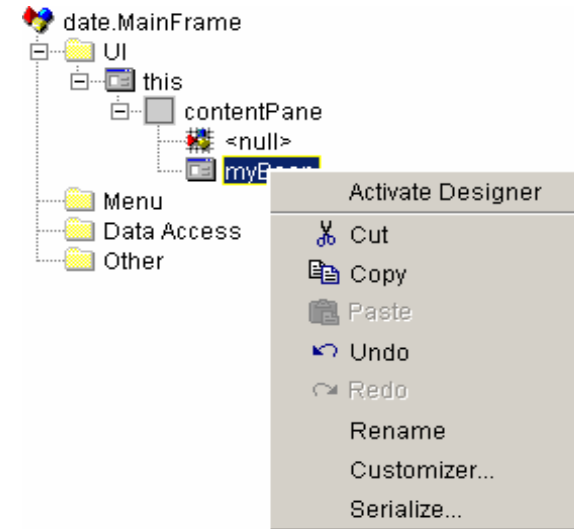
Benutzerdefinierter Editor - Interaktion



Bean-Customizer

- Customizer hat Funktion eines *Wizards*.
- Möglichkeit Bean als Gesamtes zu konfigurieren.
- Registrierung in BeanInfo

```
public class MyBeanBeanInfo extends SimpleBeanInfo {  
    public BeanDescriptor getBeanDescriptor() {  
        return new BeanDescriptor(MyBean.class,  
                                   MyBeanCustomizer.class);  
    }  
}
```



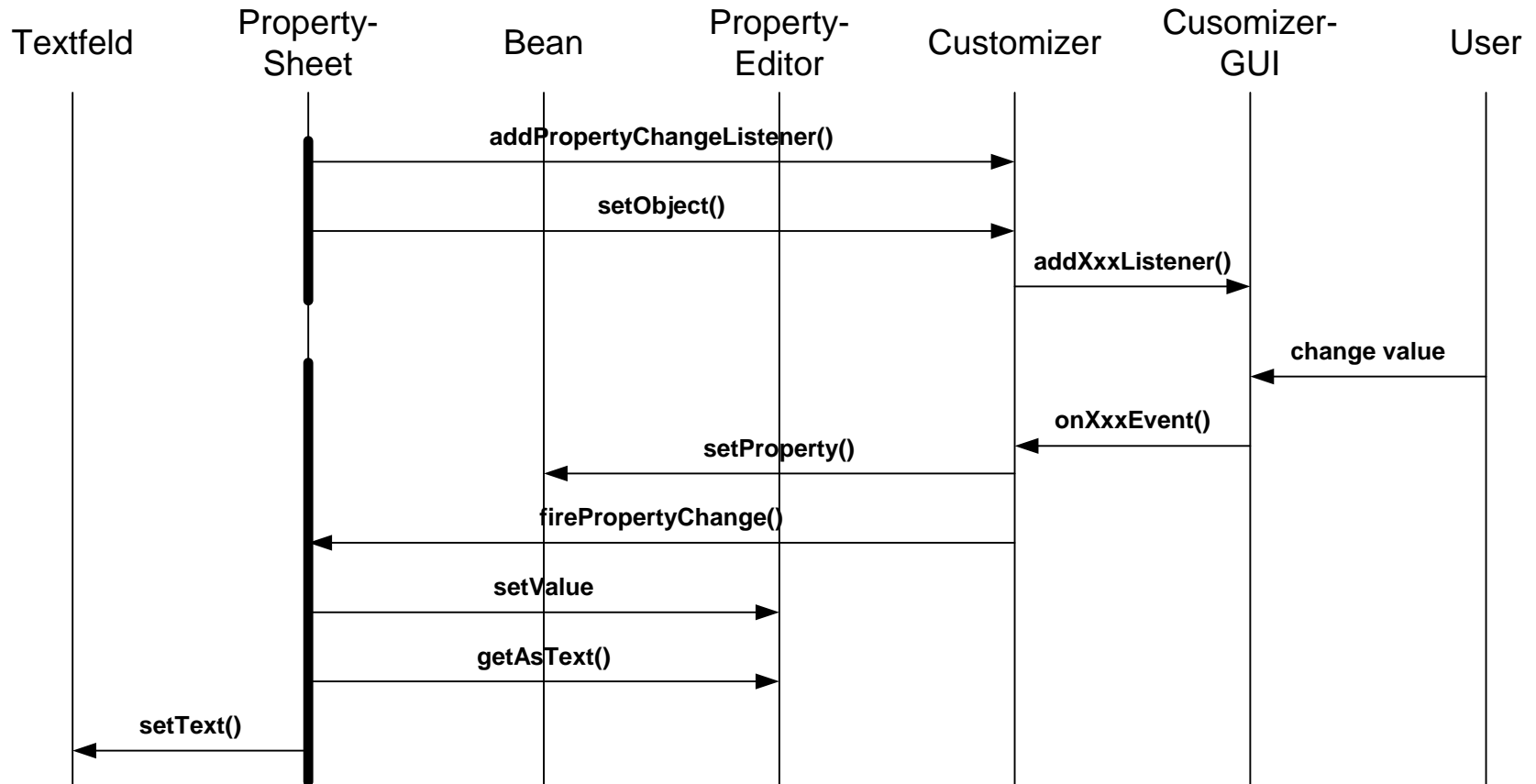
Customizer - Implementierung

- Customizer muss Customizer-Interface implementieren:

```
public interface Customizer {  
    public void setObject(Object bean);  
    public void addPropertyChangeListener(...);  
    public void removePropertyChangeListener(...);  
}
```

- `setObject`
 - Wird von Builder-Tool aufgerufen, wenn der Customizer-Dialog geöffnet werden soll.
- `add/removePropertyChangeListener`
 - Builder-Tool registriert sich beim Customizer.
 - Customizer muss mit `firePropertyChange()` Builder-Tool von Änderungen in Properties informieren.

Customizer – Interaktion



Customizer – Beispiel

```
public class MyBeanCustomizer extends Panel
    implements Customizer, ItemListener {
    protected MyBean myBean;
    protected MyProp[] propValues;
    protected PropertyChangeSupport changer =
        new PropertyChangeSupport(this);

    public void setObject(Object bean) {
        cBox = new Choice();
        cBox.addItemListener(this); add(cBox);
        myBean = (MyBean)bean;
    }

    public addPropertyChangeListener(PropertyChangeListener l) {
        changer.addPropertyChangeListener(l);
    }

    public void itemStateChanged(ItemEvent e) {
        MyProp newVal = propValues[cBox.getSelectedIndex()];
        myBean.setProperty(newVal);
        changer.firePropertyChange("myProp", null, newVal);
    }
}
```


Zusammenfassung

- Komponenten und JavaBeans
- Java Delegations-Eventmodell
 - Event-Objekt
 - Event-Quelle
 - Event-Listener
- Bean Properties
 - Simple Properties
 - Bound Properties
 - Constrained Properties
- Bean Customizing
 - Property-Editoren
 - Bean-Customizer