VUI - Das virtuelle User Interface

Wie migriert man 2 Mio LoC von Java AWT/Swing nach JavaFX?

Hochschule **Rosenheim**
University of Applied Sciences

Warum Migrieren?

- Swing Oberflächen fehlt der optische Pep. Das Look&Feel sieht i.R. angestaubt aus.
- Moderne Komponenten wie Charts, Datepicker fehlen in Swing und müssen durch Drittbibliotheken nachgerüstet werden. Viele dieser Bibliotheken werden nicht mehr weiterentwickelt.
- Die neue Generation von Programmieren lernt Swing nur noch rudimentär. Das Know-How geht langsam verloren.
- JavaFX hat eine moderne API die Nutzen aus Java 8 zieht. Viele gerade sehr populäre Muster wie z.B. das BuilderPattern sind hier umgesetzt.

Hochschule **Rosenheim**
University of Applied Sciences

Wann migriert man besser nicht?

- JavaFX benötigt mehr Ressourcen (GPU/CPU/RAM). Ohne GPU mit SW-Rendering ist JavaFX sehr langsam.
- Bei vielen Anwendungen wurde Swing so umfangreich angepasst, dass die Nachteile gering sind. Es gibt viele Beispiele für professionelle Software die mit Swing entwickelt wurde: IntelliJ, NetBeans, JProfiler, DBVisualizer
- Oracle hat gerade wenig Focus auf JavaFX und Swing. Oracle setzt gerade den Focus auf Cloud und Weboberflächen. Die Open-Source Community wird das Problem nicht lösen.

Hochschule **Rosenheim**
University of Applied Sciences

Varianten einer Migration

- Kompletter Neubau der grafischen Oberfläche
  - Für viele Anwendungen unrealistisch. Vor allem dann, wenn Anwendungslogik und UI-Logik nicht klar getrennt wurden. Viele Anwendungen werden seit den späten 90er Jahren entwickelt.
- Integration neuer JavaFX Komponenten
  - Neue Komponenten werden in JavaFX gebaut und in Swing integriert -> JFXPanel
- Migration in einen JavaFX Rahmen
  - Bestehende Swing Komponenten werden in JavaFX integriert -> SwingNode

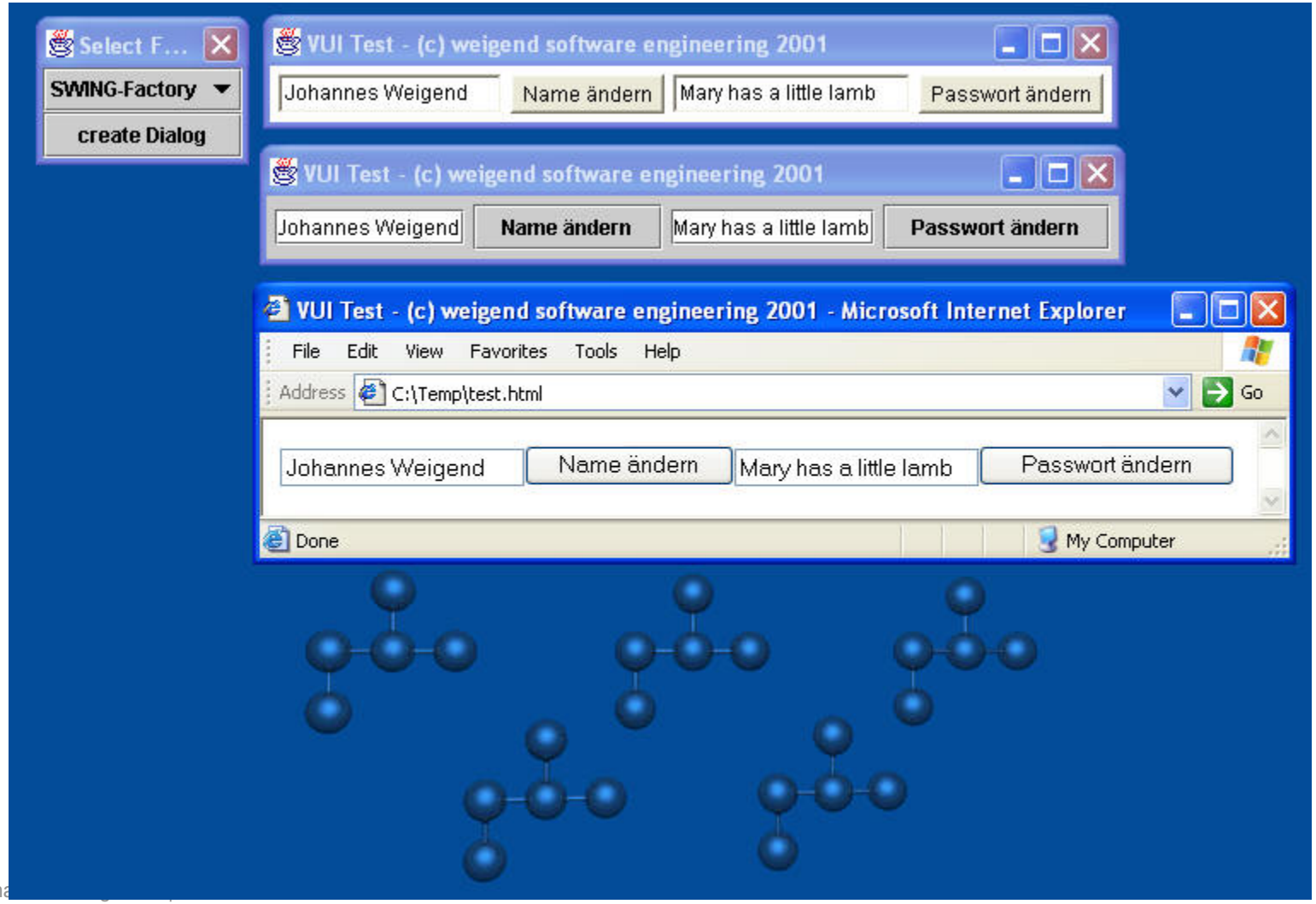
Hochschule Rosenheim
University of Applied Sciences

Umsetzungsidee: VUI

- Das VUI ist ein Adapter der die Swing-API auf JavaFX abbildet.
- Wie funktioniert das?

Hochschule Rosenheim
University of Applied Sciences

3 Varianten - 3 Implementierungen?



Hochschule Rosenheim
University of Applied Sciences

Ein einfaches Swing Programm

```
JFrame f = new JFrame();
f.setLayout(new JFlowLayout());

JButton b1 = new JButton();
JButton b2 = new JButton();

JTextField f1 = new JTextField();
JTextField f2 = new JTextField();

b1.setText("Name ändern")
b1.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    System.out.println(e);
  }
});

b2.setText("Passwort ändern");
b2.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    System.out.println(e);
  }
});
```

```
f1.setText("Johannes Weigend");
f2.setText("Mary has a little lamb");
f.setText("VUI Test");
f.add(f1);
f.add(b1);
f.add(f2);
f.add(b2);

f.pack();
f.setVisible(true);
```

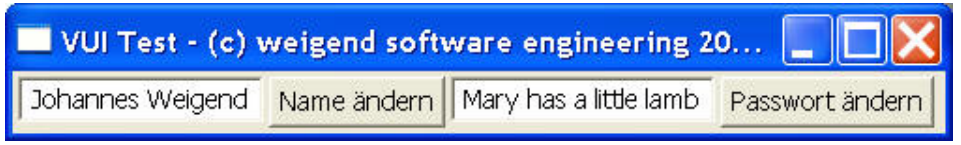
Hochschule Rosenheim
University of Applied Sciences

Ein Programm - N - Laufzeitumgebungen

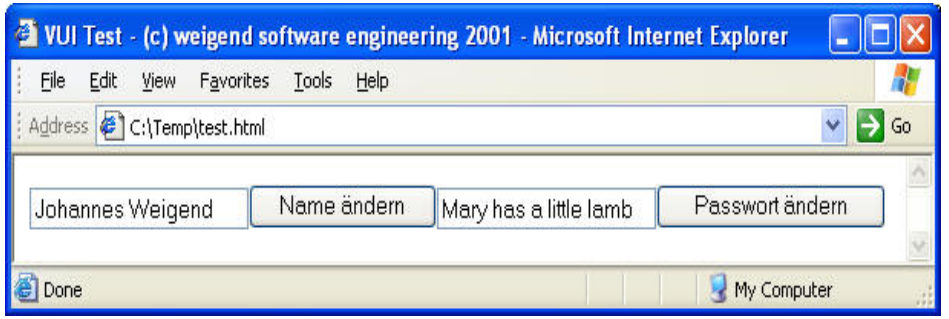
- Mit Java-AWT:

- mit Java-Swing:

- mit Eclipse-SWT:



- und mit HTML:



Hochschule Rosenheim
University of Applied Sciences

Interfaces zur Entkopplung

Die Grundidee jeglicher Entkopplung beruht auf der Verwendung von Schnittstellen (Interfaces). Ein Client darf nur das Interface und nicht die Implementierung kennen.

```
IButton button;

// Portabler UI-Code basiert ausschließlich auf IButton
button.setText();
  ...
```

Der Client-UI-Code ist vollständig entkoppelt von der Implementierung, da IButton eine Schnittstelle ist und keine Implementierung. IButton kann von von verschiedenen Klassen implementiert werden (z.B. SwingButton, AwtButton ect.). Das IButton-Interface ist minimiert auf die tatsächlich nötigen Methoden. Unnötiger Ballast wird weggelassen.

Hochschule Rosenheim
University of Applied Sciences

Verschiedene Implementierungen einer Schnittstelle

Das IButton-Interface wird von je einer Klasse pro unterstützter API implementiert:

```
public class JFXButton   implements IButton { …}
public class HTMLButton  implements IButton { …}
public class SwingButton implements IButton { …}
```

Je ähnlicher die IButton-Schnittstelle den zur Verfügung stehenden Implementierungen ist, umso einfacher lassen sich Adapter ohne Aufwand programmieren:

```
public class SwingButton extends JButton implements IButton{
        // nothing to be done !!!
}
```

Hochschule Rosenheim
University of Applied Sciences

Fabriken entkoppeln die Instanzierung

Um verschiedene Implementierungen einer Schnittstelle verwenden zu können wird eine Factory benutzt welche die Objekterzeugung übernimmt:

// nicht möglich !!! IButton button = new SwingButton();

```
IFactory factory = ...; // siehe VFactory

// SwingButton/JFXButton/HtmlButton
IButton button = factory.createButton();
button.setText("Ok"); …
```

Hochschule Rosenheim
University of Applied Sciences

Das Abstract Factory Muster

Die Factory entkoppelt den Anwendungscode von der konkreten Objekterstellung. Ein solches Vorgehen ist als "Abstract Factory Pattern" bekannt (Vgl. Gamma - Abstract Factory). Die Factory ist selbst eine Schnittstelle. Konkrete Implementierungen gibt es für alle zu unterstützenden APIs. Jede konkrete Factory legt nur Objekte "ihres" Typs an.

```
                HtmlFactory
                JFXFactory
public class SwingFactory implements IFactory
{
    public IButton createButton()
    {
                    HtmlButton();
                    JFXButton();
        return new SwingButton();
    }
    ...
}
```


Hochschule Rosenheim
University of Applied Sciences

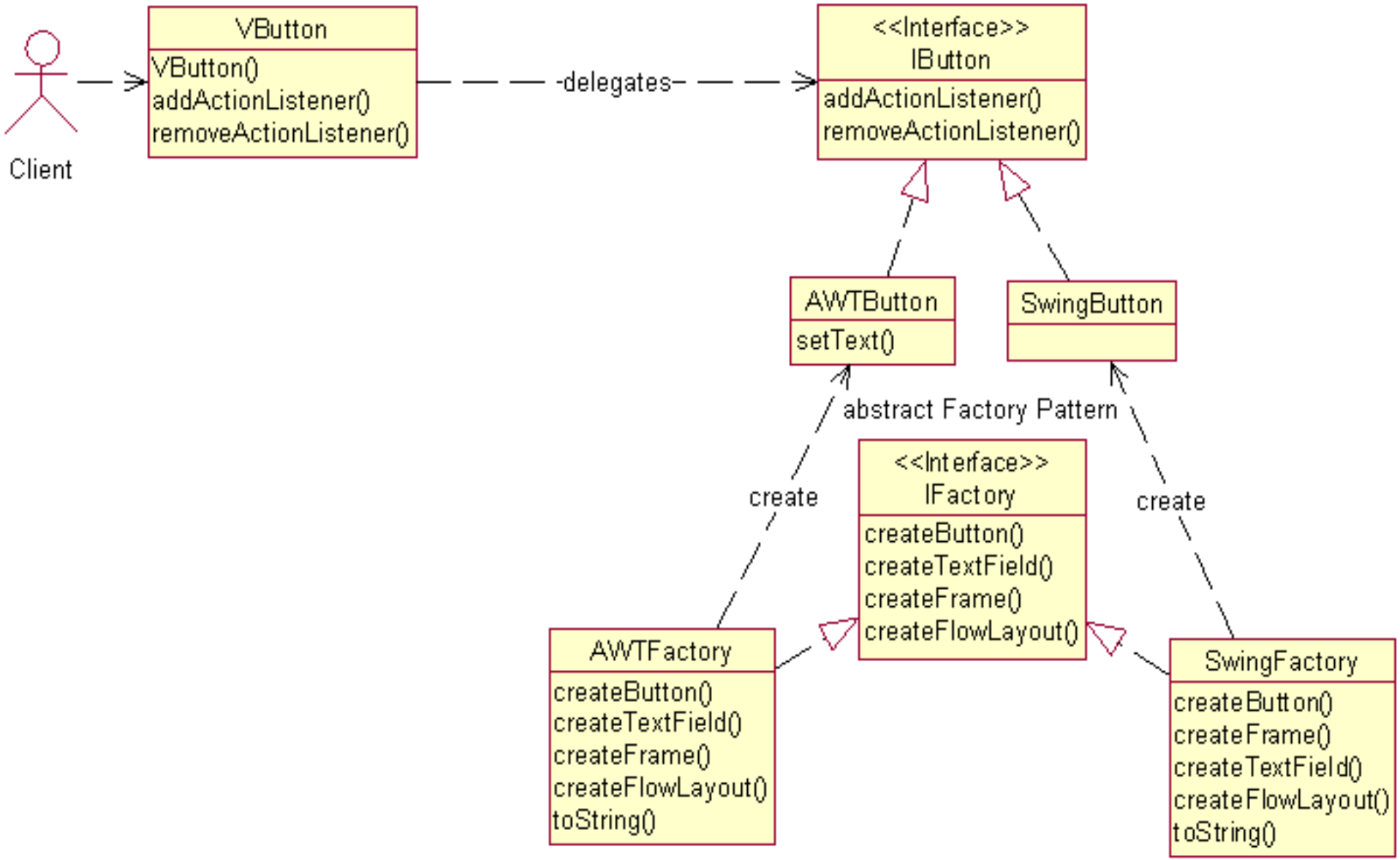
Wrapperklassen verstecken die Fabrik.

Die Erzeugung über eine Factory ist unschön und entspricht nicht dem üblichen Programmiermodel. Daher ist es sinnvoll eine Hilfsklasse zu bauen, welche die Objekterzeugung mithilfe einer Factory vor dem Client versteckt und eventuell eine Schnittstellen- oder Hierarchieanpassung durchführen kann.

```
Beispiel:
// generierbar !
class VButton implements IButton
{
    VButton()
    {
        delegate = VFactory.getInstance().createButton();
    }

    public void setText(String text)
    {
        delegate.setText(text);
    }
    IButton delegate; // TIE / Bridge-Pattern (siehe Hierarchie)
}
```

Hochschule Rosenheim
University of Applied Sciences

VUI-Prinzip


VButton
VButton()
addActionListener()
removeActionListener()
Client
-delegates-
<<Interface>>
IButton
addActionListener()
removeActionListener()
AWTButton
setText()
SwingButton
abstract Factory Pattern
create
create
<<Interface>>
IFactory
createButton()
createTextField()
createFrame()
createFlowLayout()
AWTFactory
createButton()
createTextField()
createFrame()
createFlowLayout()
toString()
SwingFactory
createButton()
createFrame()
createTextField()
createFlowLayout()
toString()

Singleton – VFactory

Die VFactory ist ein Singleton und dient als globale Factory-Auswahlstelle. Dort kann die zu verwendende Factory gesetzt werden

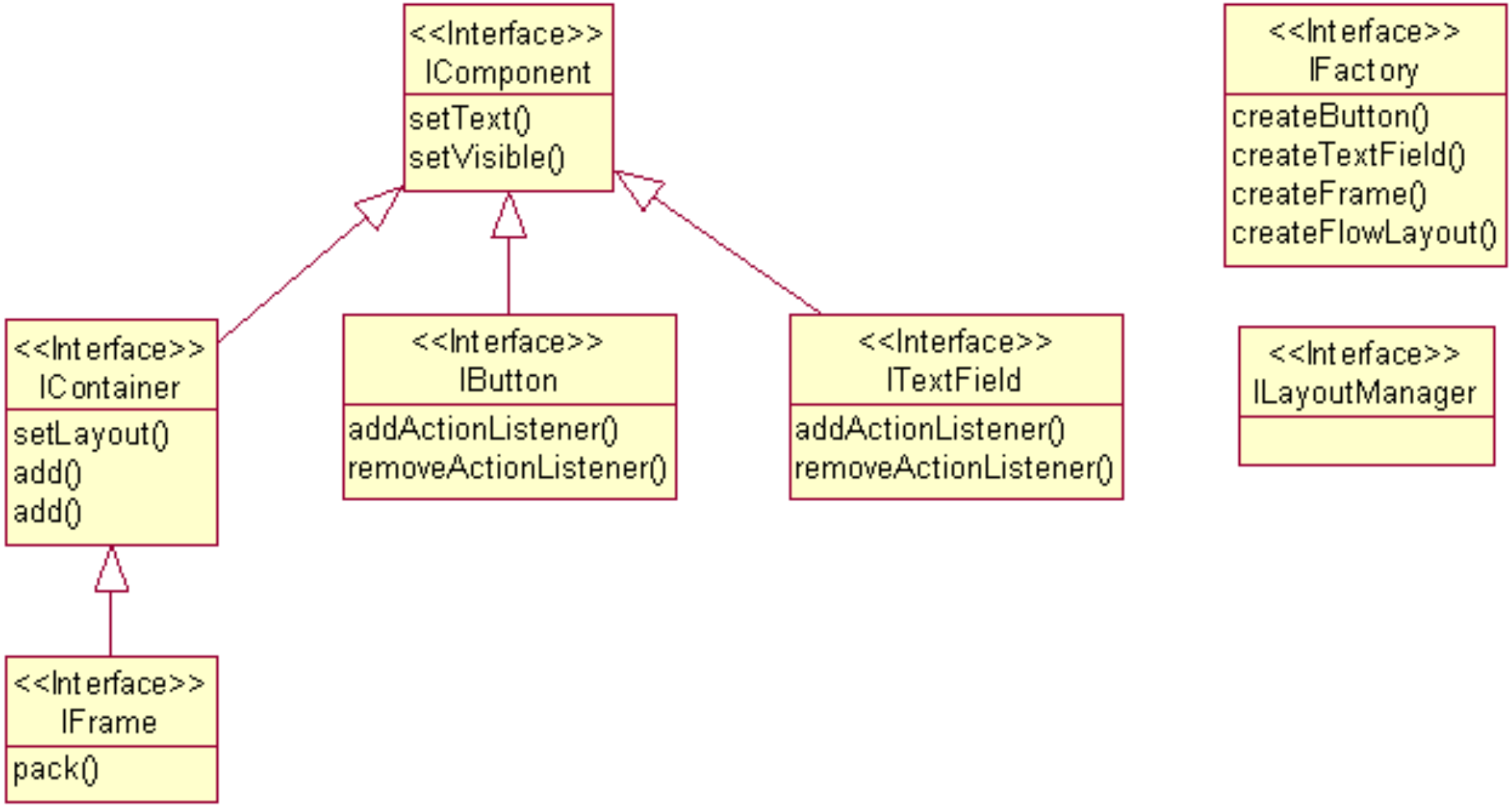
```
public class VFactory implements IFactory
    {
        private VFactory()
        {
            this.factory = new vui.swing.SwingFactory(); // DEFAU
        }

        public IButton createButton()
        {
            return factory.createButton();
        }
...

        public static VFactory getInstance()
        {
            return instance == null ? instance = new VFactory() :
        }

        public void setFactory(IFactory factory)
        {
            this.factory = factory;
        }

        private IFactory factory;
        private static VFactory instance;
    }
```

Hochschule Rosenheim
University of Applied Sciences

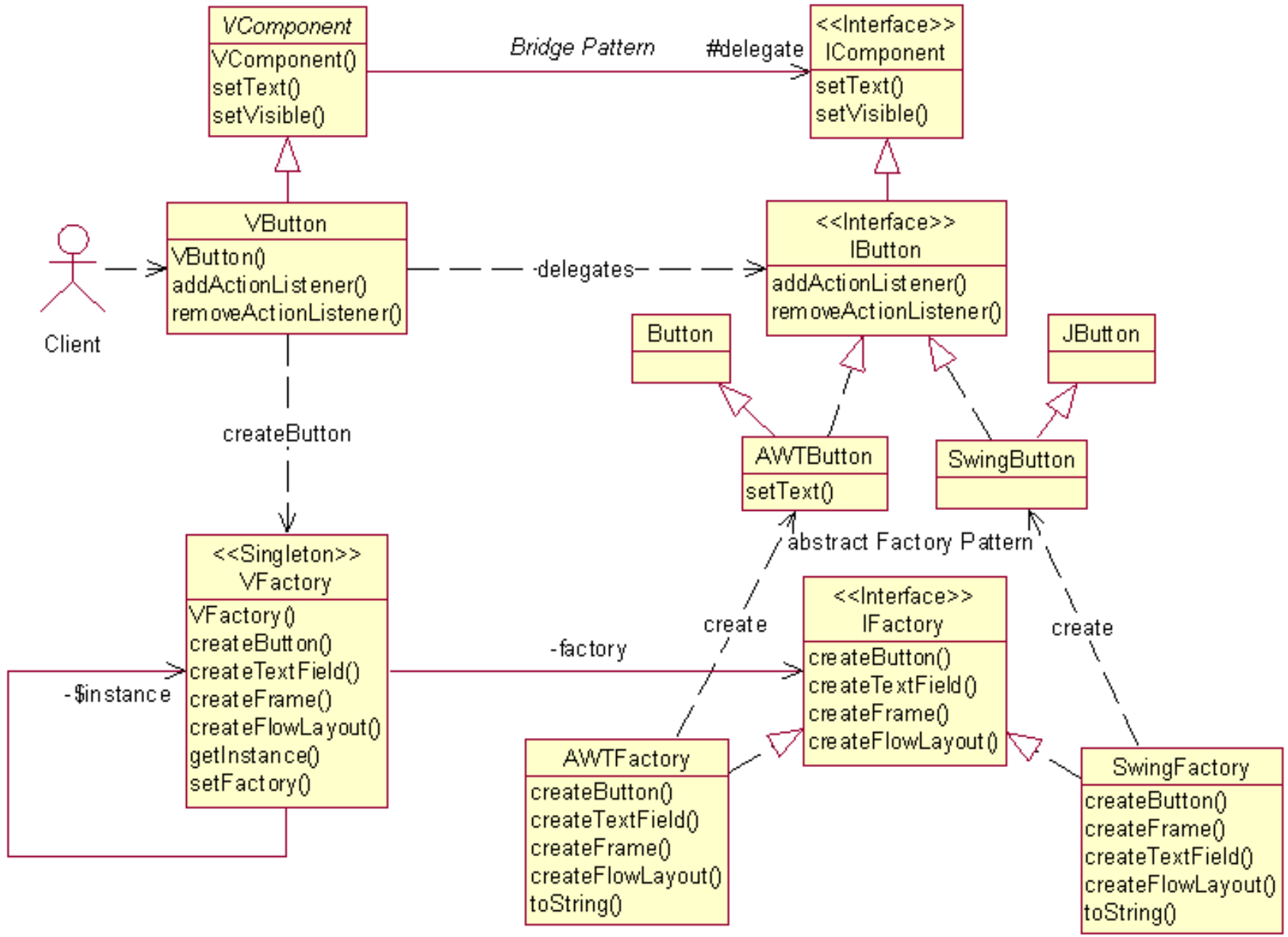
Schnittstellen Hierarchie

- Damit verschiedene Bedienelemente gleich behandelt werden können muß es eine Schnittstellen-Hierarchie geben. Für alle zu unterstützenden Objekte einer konkreten API müssen eigene Schnittstellen bereitgestellt werden. Je ähnlicher die Hierarchie der zu unterstützenden APIs ist, desto weniger Aufwand ergibt sich bei der Implementierung.

Hochschule Rosenheim
University of Applied Sciences


<<Interface>>
IComponent
setText()
setVisible()
<<Interface>>
IFactory
createButton()
createTextField()
createFrame()
createFlowLayout()
<<Interface>>
IContainer
setLayout()
add()
add()
<<Interface>>
IButton
addActionListener()
removeActionListener()
<<Interface>>
ITextField
addActionListener()
removeActionListener()
<<Interface>>
ILayoutManager
<<Interface>>
IFrame
pack()

VUI-Implementierung



Migration nach JavaFX

- Teile einer Swing Anwendung können per VUI Adapter entkoppelt werden (z.B. die interne Fensterverwaltung einer Anwendung)
- Im einfachsten Fall müssen im Swing Code nur Import-Statements angepasst werden
  - (de.fhro.vui.JButton anstelle von javax.swing.JButton)
- Ein VUI wird niemals die gesamte Swing API abdecken können
- Damit kann ein gesamte Anwendung ohne größere Anpassungen auf eine neue Technologie umgestellt werden.

Hochschule **Rosenheim**
University of Applied Sciences