

Medientechnik

Übung

Heute

- Java Media Framework (JMF):
 - Grundlagen
 - Einfacher Player
 - Plugins

Java Media Framework

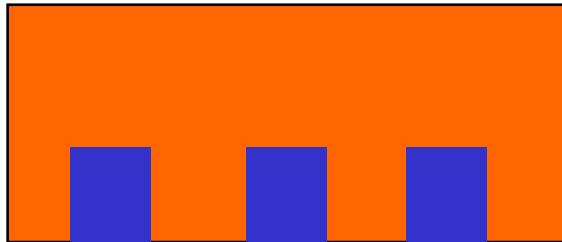
- Kostenloses, plattformunabhängiges und objektorientiertes Medien-Framework
- Hauptfunktionen:
 - Abspielen von Medien
 - Verarbeitung von Mediendaten in Echtzeit
 - Erfassung von Datenströmen
 - Speichern von Mediendaten
 - Strombasierte Übertragung (*streaming*) von Mediendaten
- JMF *nicht* Bestandteil der Standard-Java2-Distribution
 - Cross-Platform-Implementierung und „Performance Packs“

Frameworks

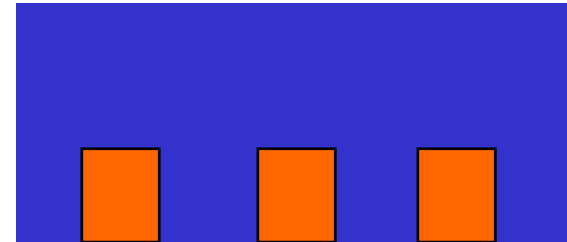
- **Definition** (Taligent): „A *framework* is a set of prefabricated software building blocks that programmers can use, extend, or customize for specific computing solutions.“
- **Definition** (nach Pomberger/Blaschek): Ein "*framework*" (Rahmenwerk, Anwendungsgerüst) ist eine Menge von zusammengehörigen Klassen, die einen abstrakten Entwurf für eine Problemfamilie darstellen.
- Ziele:
 - Wiederverwendung von Code, Architektur und Entwurfsprinzipien
 - Wiederverwendung von Verhaltensschemata einer Gruppe von Klassen
 - Homogenität unter verschiedenen speziellen Anwendungssystemen für eine Problemfamilie (z.B. ähnliche, ergonomische Bedienung)

Vergleich Klassenbibliothek-Framework

Klassenbibliothek



Framework



 Anwendungsspezifische Teile
 Vorgefertigte Teile

***"Don't call us,
we call you"***

(„Hollywood-Prinzip“)

Anpassung
nur durch Instanziierung

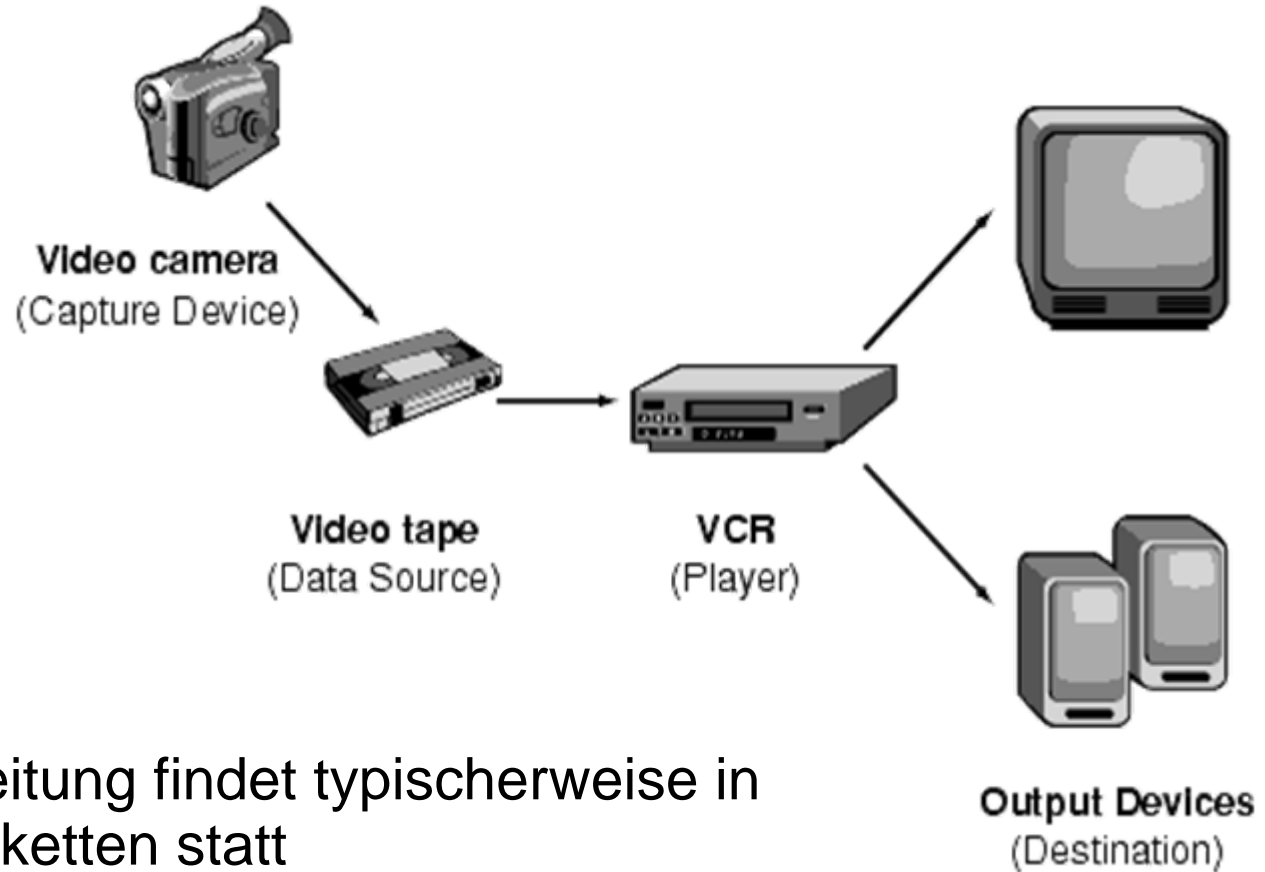
Anpassung
auch durch Spezialisierung

Ablaufsteuerung
nicht vordefiniert

Ablaufsteuerung
im wesentlichen vordefiniert

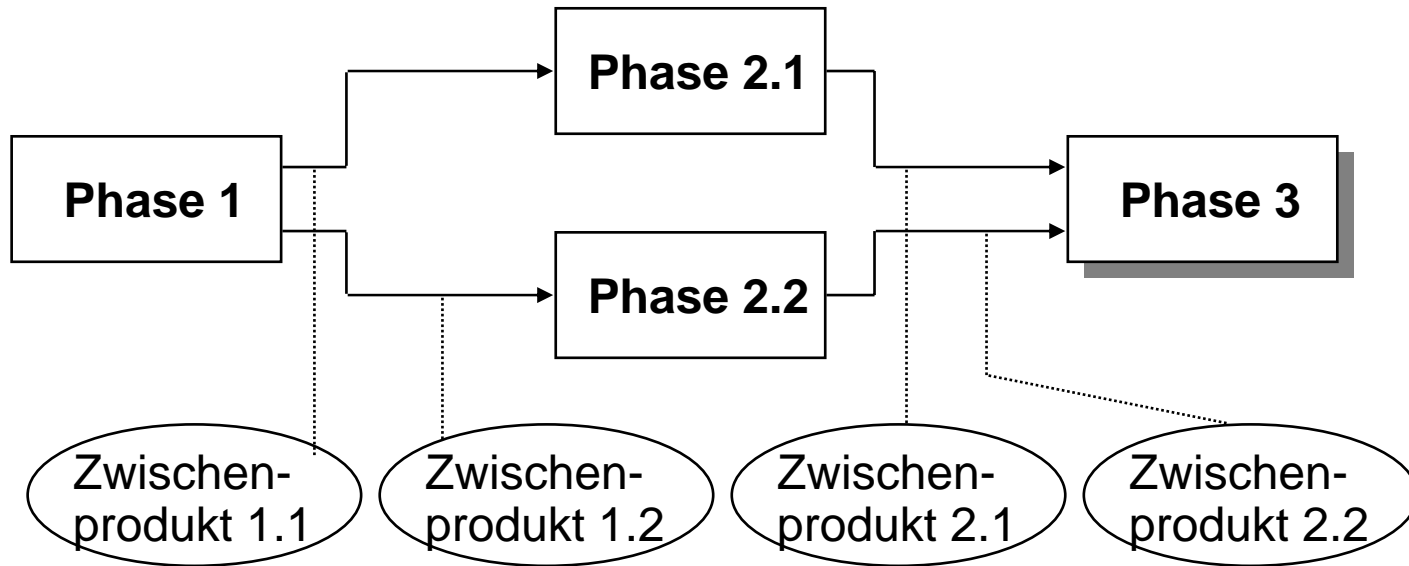
Die Grenze ist fließend, siehe z.B. Java AWT !

Verarbeitungsketten in der Medienbearbeitung



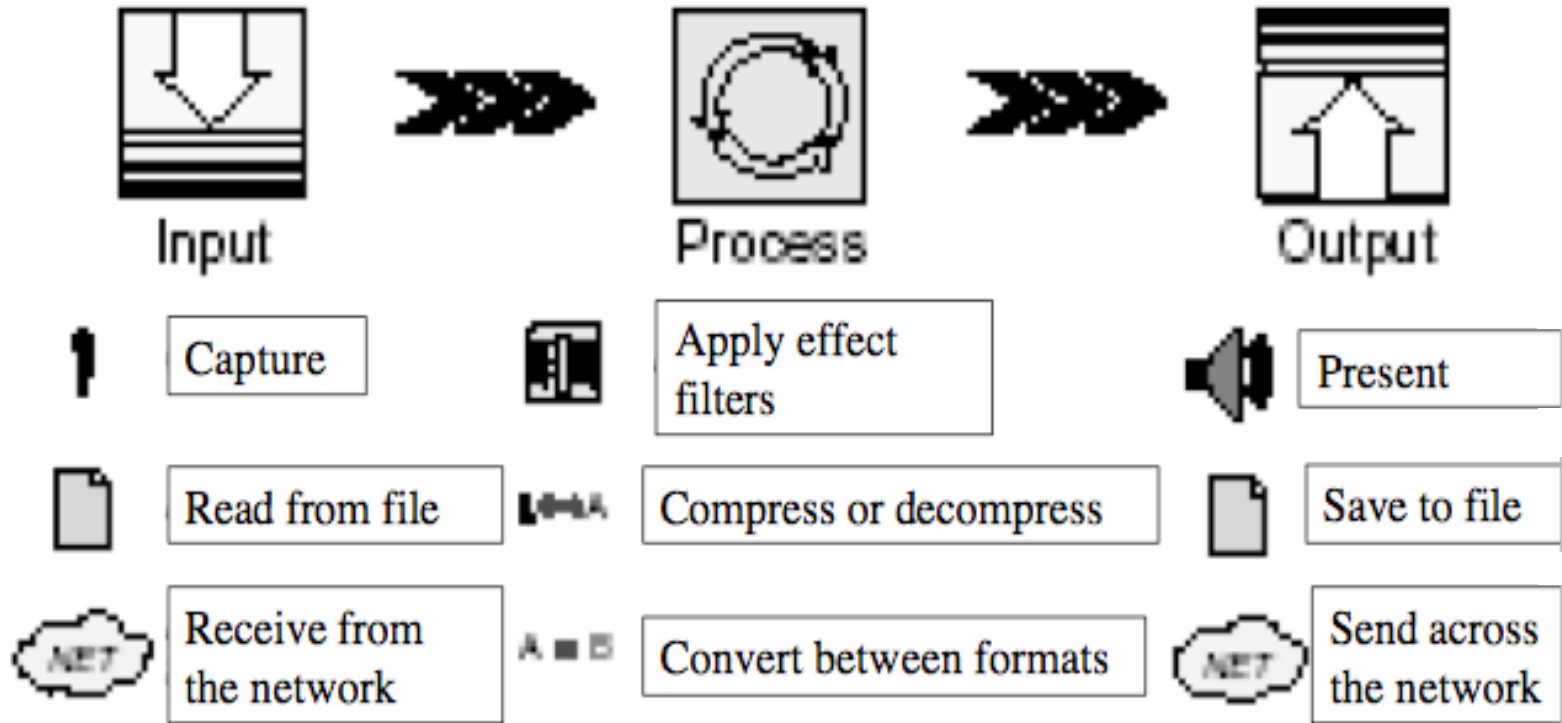
- Medienbearbeitung findet typischerweise in Verarbeitungsketten statt
 - Medien als Container repräsentiert („offline“):
Lange Bearbeitungszeiten, grosse Dateneinheiten
 - Medien als Ströme repräsentiert („online“):
Kurze Bearbeitungszeiten, kleine Dateneinheiten

Architekturmuster "Kette"



- Inkrementelle oder phasenweise Verarbeitung
- Anderer Name: Pipes & Filters
- Beispiele:
 - UNIX pipes
 - Batch-sequentielle Systeme
 - Compiler-Grundstruktur

Verarbeitungsketten-Modell in JMF



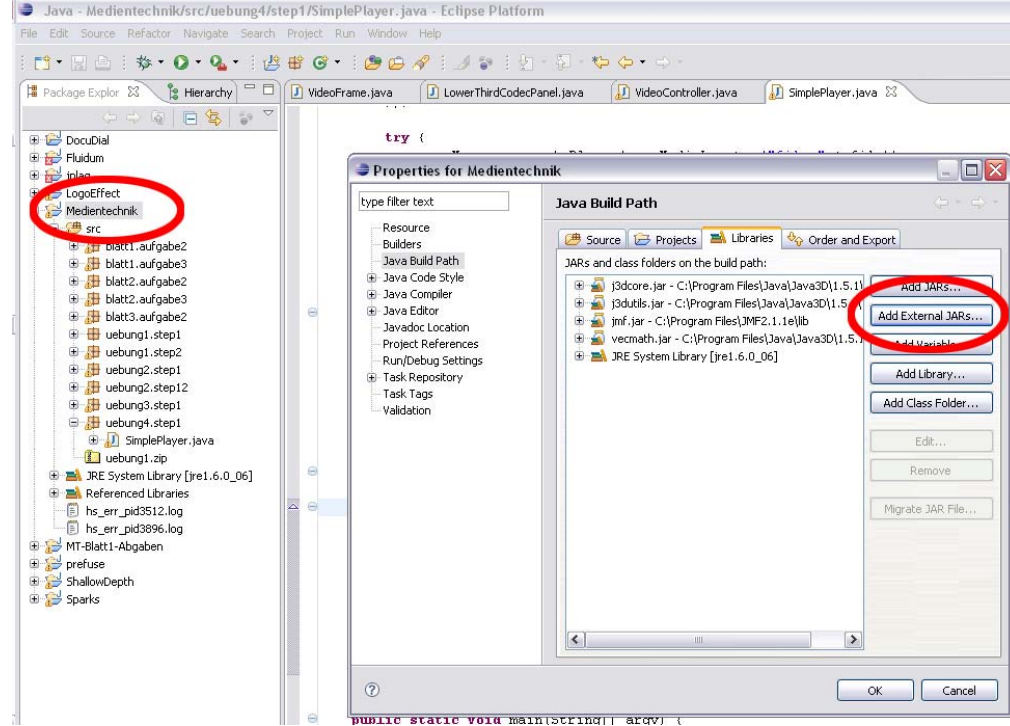
Prinzipiell ist jede Kombination der möglichen Eingabeoptionen, Verarbeitungsschritte und Ausgabeoptionen möglich

Beispielcode

`/home/proj/mi_mt_ss08/jmf/src`

Bitte alle java-Dateien in euer home-
Verzeichnis kopieren!

CIP-Pool JMF



- Rechtsklick Projekt -> "Properties"
- links: "Java Build Path"
- Button "Add External JARs..."
- **jmf.jar** aus
- `/soft/IFI/lang/j2sdk-1.4.2_03-sun/iX86-unknown-linux/jre/lib/ext`

Einfacher Player

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.media.*;

public class SimplePlayer extends JFrame {

    private Player p = null;
    private SimplePlayer f = this;

    public SimplePlayer(String file) {
        setTitle("Simple JMF Player");
        getContentPane().setLayout(new BorderLayout());
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent event) {
                p.stop();
                p.deallocate();
                System.exit(0);
            }
        });
    }
}
```

SimplePlayer.java

Einfacher Player (2)

```
try {  
    p = Manager.createPlayer(new MediaLocator("file:" + file));  
    p.addControllerListener(new ContrEventHandler());  
    p.realize();  
} catch (Exception e) {  
    System.out.println("Exception " + e);  
    System.exit(-1);  
}  
}  
  
public void addControlPanel() {  
    getContentPane().add(p.getControlPanelComponent(), BorderLayout.SOUTH);  
    Component vc = p.getVisualComponent();  
    if (vc != null)  
        getContentPane().add(vc, BorderLayout.NORTH);  
    setSize(400, 300);  
    setVisible(true);  
}
```

Einfacher Player (3)

```
class ContrEventHandler implements ControllerListener {  
  
    public synchronized void controllerUpdate(ControllerEvent e) {  
        if (e instanceof RealizeCompleteEvent) {  
            f.addControlPanel();  
            p.start();  
        } else if (e instanceof EndOfMediaEvent) {  
            p.stop();  
            p.setMediaTime(new Time(0));  
            p.start();  
        }  
    }  
}  
  
public static void main(String[] argv) {  
    String file = "/home/proj/mi_mt_ss08/jmf/source.avi";  
    SimplePlayer pf = new SimplePlayer(file);  
}
```

Packungsgrad von Medien

- **SourceStream:**

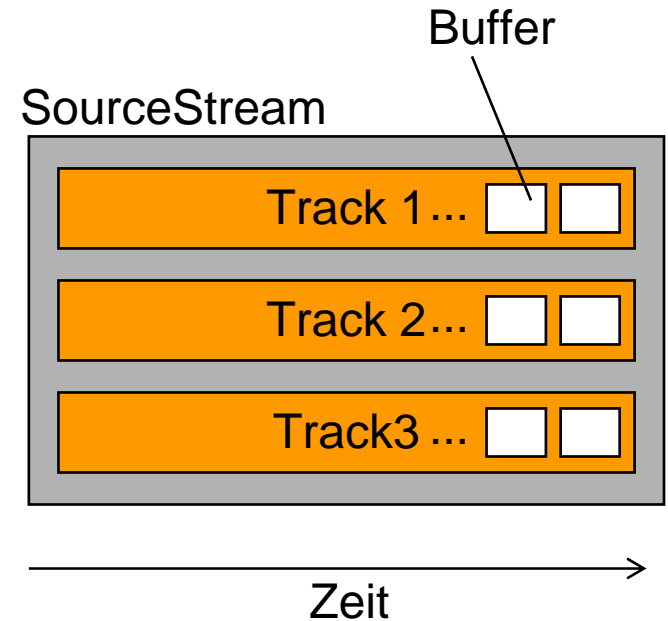
- Kapselt Medium
- Beschrieben durch **ContentDescriptor**

- **Track:**

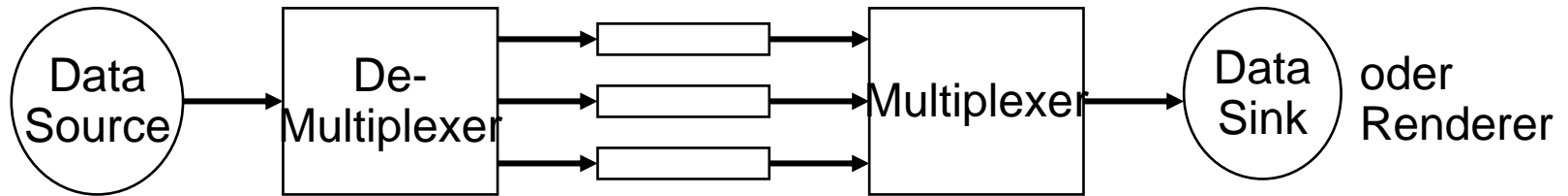
- Einzelkomponente eines Stroms (z.B. Video-, Audiospur)
- Zugriff auf Mediendaten

- **Buffer:**

- Einzelner Datenblock eines **Track**
- Wird zur Weitergabe von Daten in Verarbeitungsketten genutzt
- Detaillierte Beschreibung: **Format-Objekt**
 - **AudioFormat**
 - **VideoFormat (RGB, YUV, JPEG, ...)**



Verarbeitungsketten in JMF

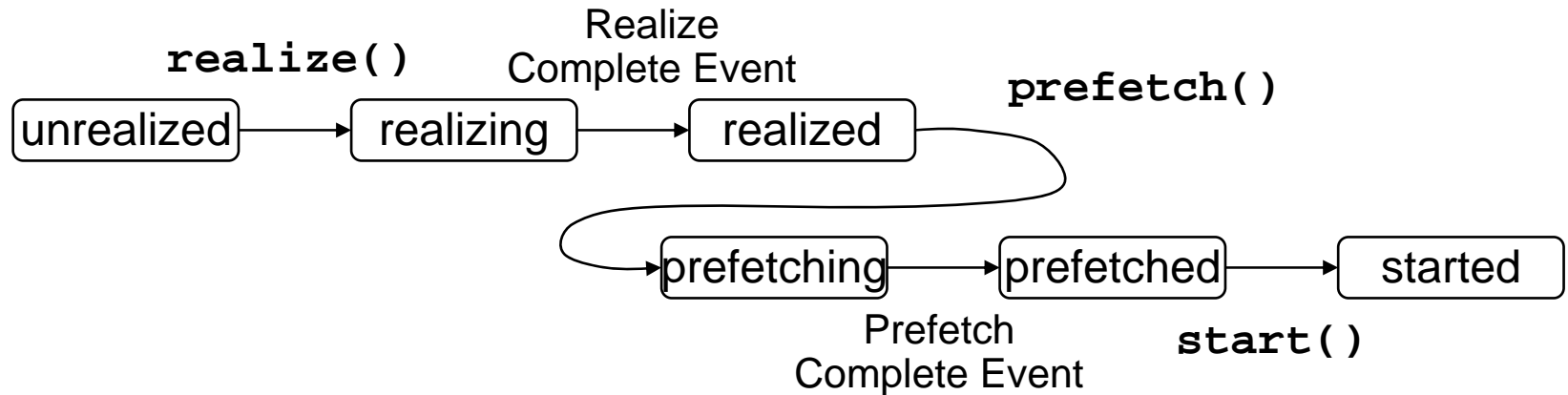


- Kette von Transformatoren in JMF:
 - Echtzeitanforderung: Bei Zeitüberschreitung wird **Buffer** gelöscht
 - Alle Transformatoren passiv
 - Wer steuert die ganze Kette (und gibt z.B. den Zeittakt vor)?
- **Controller**-Schnittstelle
 - Schnittstelle für Steuerung von Verarbeitungsketten
 - Spezialfälle:
 - **Processor**: Allgemeine Verarbeitungskette
 - **Player**: Einfacher Fall mit trivialer Transformation und Rendering
 - **DataSink**: Dateiausgabe, ähnlich aber nicht von **Controller** abgeleitet

Manager-Klassen in JMF

- *Manager*
 - zur Verwaltung und Konstruktion von Controller-Objekten (*Player*-, *Processor*-, *DataSource*-, und *DataSink*- Objekte)
Beispiel: `Manager.createPlayer(DataSource d);`
- *PackageManager* (Verwaltung aller Pakete, die die JMF-Dateien enthalten)
- *CaptureDeviceManager* (Verwaltung aller vorhandenen Eingabegeräte)
- *PluginManager* (Verwaltung aller verfügbaren Plug-Ins, z.B. Multiplexer, Codecs)

Zustandsmodell von Player

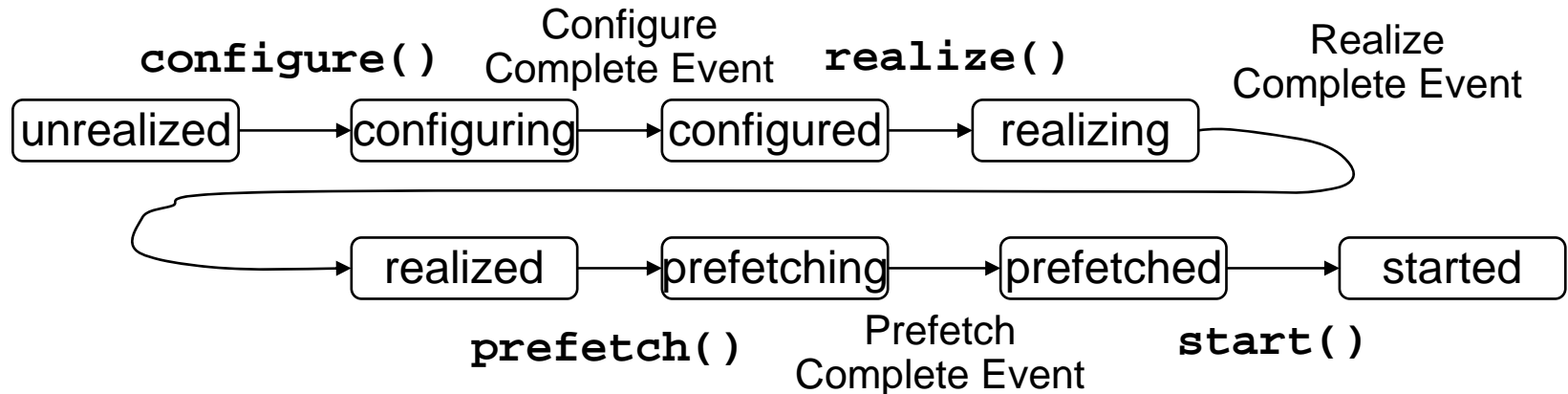


- *Unrealized:*
 - Anfangszustand
- *Realizing:*
 - Medienabhängige Teile des Players werden bereitgestellt
- *Prefetching:*
 - Eingabestrom wird soweit gelesen wie nötig, um Puffer zu füllen
- *Started:*
 - Verarbeitung läuft

Processor

- **Processor** ist die Abstraktion einer medienverarbeitenden Einheit
- Funktionsmöglichkeiten:
 - Nimmt Datenquellen als Eingabe
 - Führt beliebige benutzerdefinierte Transformationen aus
 - Liefert bearbeitete Daten ab
 - Auf Ausgabegerät (Rendering, analog Player)
 - Als **DataSource**
- Vom **Processor** gelieferte **DataSource** kann weiterverarbeitet werden
 - In einem weiteren **Processor**
 - In einer **DataSink** (z.B. Speicherung in einer Datei)
- Wichtigster Unterschied zu **Player**:
 - Separate Bearbeitung verschiedener Tracks der Quelle
- **Processor** wird erzeugt mit **Manager.createProcessor (DataSource ds)**

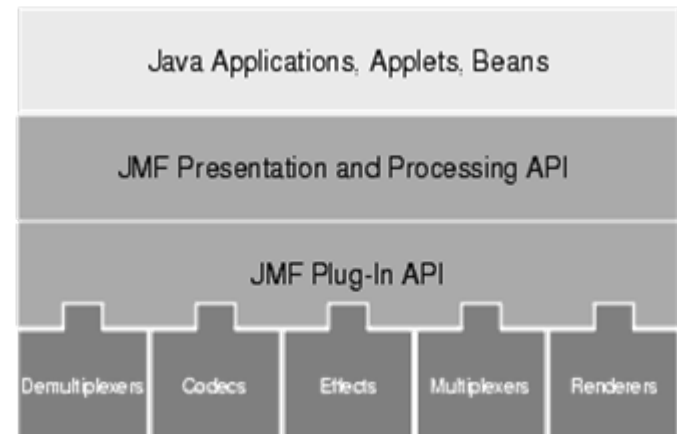
Zustandsmodell von Processor



- *Configuring:*
 - Die Eingabe wird auf die enthaltenen Medien (Spuren, *tracks*) analysiert
- *Configured:*
 - Bearbeitung für die einzelnen Spuren kann separat definiert werden
- *Realizing:*
 - Wie beim Player: Verarbeitungskette wird abhängig von den konkreten Mediendaten bereitgestellt
- *(alles andere wie beim Player)*

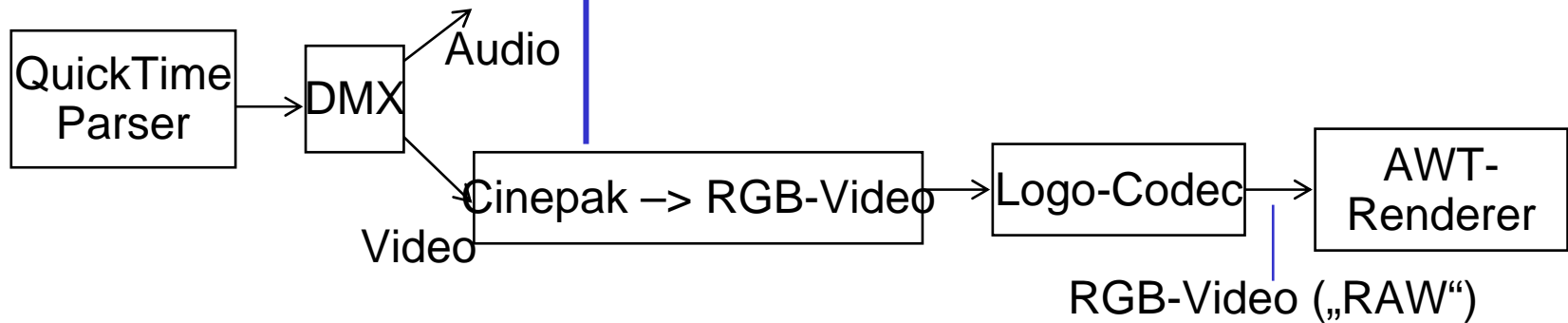
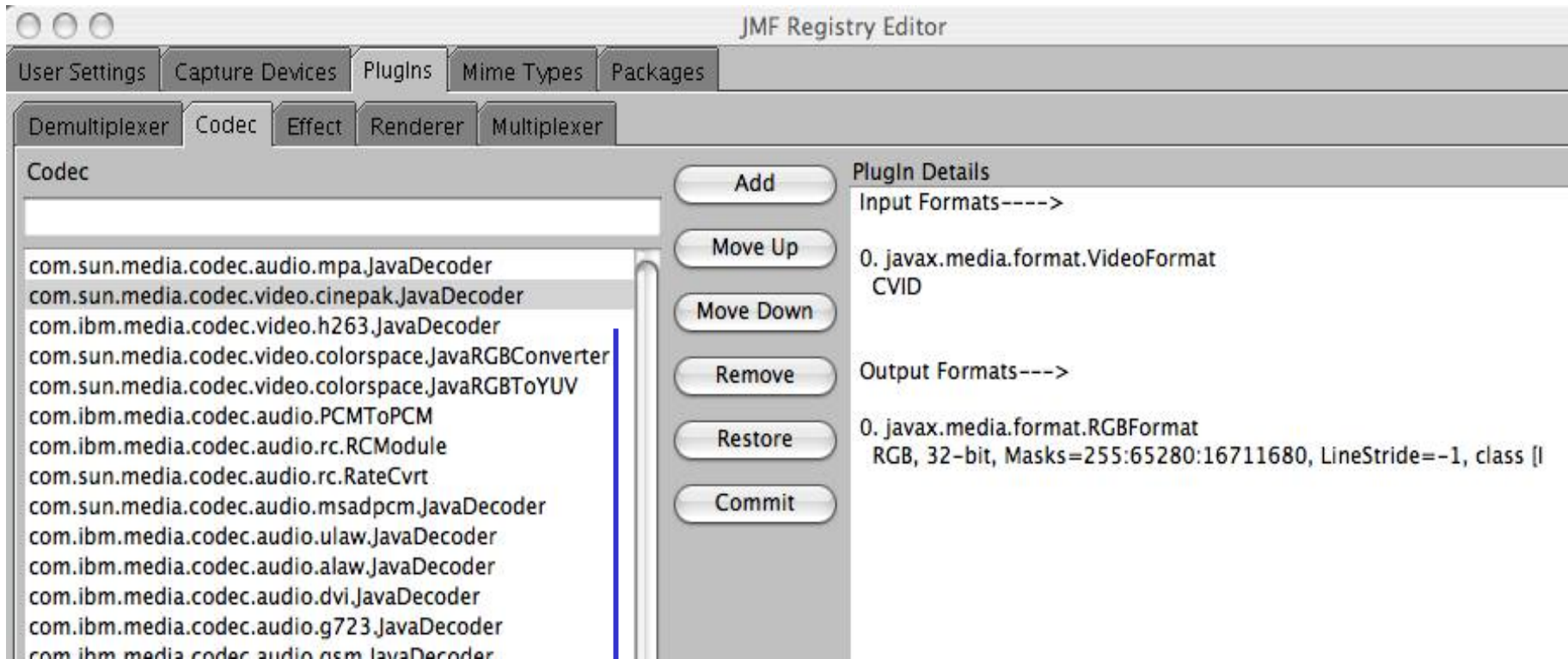
Plugin

- Basisschnittstelle für alle Elemente der Verarbeitungskette, die Daten in einem bestimmten Format ein- und/oder ausgeben
- Unterschnittstellen:
 - **Codec**
 - Unterschnittstelle **Effect**
 - **Multiplexer, Demultiplexer**
 - **Renderer, VideoRenderer**
- **Plugin-Schnittstelle** unterstützt allgemeine Operationen **open()**, **close()**, **reset()**, die bei Konfiguration aufgerufen werden
 - Weitere Details in den Unterschnittstellen definiert



Beispiel

Verarbeitungskette für Logo-Effekt



Implementierung eines eigenen Codec/Effect

- Implementierung der Schnittstelle `Codec` bzw. ihrer trivialen Unterschnittstelle `Effect`
 - Selbstbeschreibung bezüglich Strom-Datenformaten
 - Realisierung der Konfigurations-Operationen `open()`, `close()`, `reset()`
 - Fachlicher Kern: `process()`
- Implementierung mindestens eines `Controls` zur Steuerung

```
public interface Codec { // Auszug!
    Format[] getSupportedInputFormats()
    Format[] getSupportedOutputFormats()
    int process(Buffer input, Buffer output)
    ...
    Object getControl(String controlType)
    ...
}
```

Control

- **Control:**
 - Basisschnittstelle für alle Funktionen, die die Verarbeitung steuern
- ```
public interface Control {
 java.awt.Component getControlComponent()
}
```
- `getControlComponent` liefert eine Swing-Komponente, die als GUI der Steuerung dient, kann `null` sein, wenn kein GUI vorhanden
  - Jedes Plugin (also auch jeder Codec) muss anbieten:  
`Object getControl (String classname)`
    - Das Resultat-Objekt muss `Control` implementieren und `classname`-Objekte steuern können
  - Fachliche Funktionen für das spezifische Plugin (meist Setzen von Einstellungen) in der Steuerungsschnittstelle



# Code