# Java Media Framework

*„Capturing, processing, and presenting time-based media with JAVA.“*

# Java Media Framework

- Definition:

  Java Media Framework (kurz: JMF) ist eine API für die Integration von zeitabhängigen Medien in Java-Applets und -Applikationen.

- JMF unterstützt:
  - Erfassung (Capture)
  - Verarbeitung (Processing)
  - Präsentation (Presentation)

  ...von zeitabhängigen Medien.

# Übersicht

- JMF-Architektur
- Präsentation von Mediendaten
- Verarbeitung von Mediendaten
- Erfassung von Mediendaten
- Daten-Speicherung und -Übertragung
- Erweiterbarkeit des JMF
- Aufgabe: JMF Terminal
- Links und Hinweise

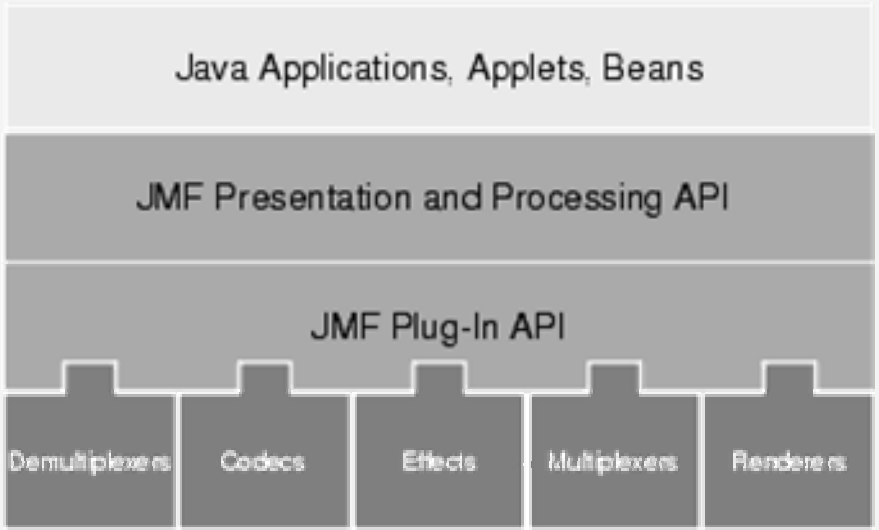
# JMF-Architektur (1)

- Unterstützte Dateiformate:

| Videoformate | Audioformate |
|---|---|
| AVI | AIFF |
| MPEG | GSM |
| Quicktime | WAVE |
| | Sun Audio |
| | MIDI |
| | RMF |

# JMF-Architektur (2)

- High-Level & Low-Level Architektur:
  - High-Level API: Erfassung, Präsentation, und Bearbeitung zeitabhängiger Medien
  - Low-Level API: Plug-Ins (Erweiterungen)



# JMF-Architektur (3)

- Basismodell:

**Eingabegeräte**
(Capture Devices)

**Datenquelle**
(Data Source)

**Player**

**Ausgabegeräte**
(Output Devices)

# JMF-Architektur (4)

- Das Zeitmodell:
  - Verfolgung der Zeit für einen „einzelnen“ Medienstrom durch Grundfunktionen für Timing und Synchronisation:

    *Clock*-Interface:
    - *TimeBase* (konstant tickende Zeitquelle)
    - *MediaTime* (aktuelle Position im Medienstrom)
    - *Rate* (Playback Rate)
    - Zusätzlich: *Duration*-Interface

    Formel:

    *MediaTime = MediaStartTime + Rate(TimeBaseTime - TimeBaseStartTime)*

# JMF-Architektur (5)

- Die Manager:
  - Vermittlerobjekte zwischen den zahlreichen Interfaces und Implementierungen davon:
    - *Manager* (zur Konstruktion von *Player*-, *Processor*-, *DataSource*-, und *DataSink*- Objekten)
      Bsp: *Manager.createPlayer(dataSource);*
    - *PackageManager* (Register aller Pakete, die die JMF-Dateien enthalten)
    - *CaptureDeviceManager* (Register aller vorhandenen Eingabegeräte)
    - *PlugInManager* (Register aller verfügbaren Plug-Ins, z.B. Multiplexer, Codecs)

# JMF-Architektur (6)

- Das Ereignismodell:
  - Strukturierter Eventmechanismus (Lieferung von Infos über den aktuellen Zustand des Mediensystems; Möglichkeit, auf medien-bezogene Fehler zu reagieren)
  - *MediaEvent* (Subklassen: *GainChangeEvent*, *DataSinkEvent*, *ControllerEvent*, etc.)
  - Implementieren eines *Listener*-Interfaces & Registrieren des eventauslösenden Objektes durch den Aufruf von *addListener(),* z.B. player.addControllerListener(listener);

# JMF-Architektur (7)

- Das Datenmodell:
  - *DataSource* (Datenquelle):
    - Enthält Information über Location sowie Protokoll und Software von Medien
      - Location: *URL* oder *MediaLocator*
      - Protokoll: HTTP, FILE, RTP
    - Dient der Transferverwaltung
    - Eine *DataSource* kann nicht wiederverwendet werden! (Eine DataSource pro Stream)
    - Kategorisierung nach der Art der Transfereinleitung:
      - Pull-DataSource (Client startet Transfer, z.B. HTTP, FILE)
      - Push-DataSource (Server startet Transfer, z.B. RTP)
    - Besondere DataSources: *CloneableDataSource*, *MergingDataSource*

# JMF-Architektur (8)

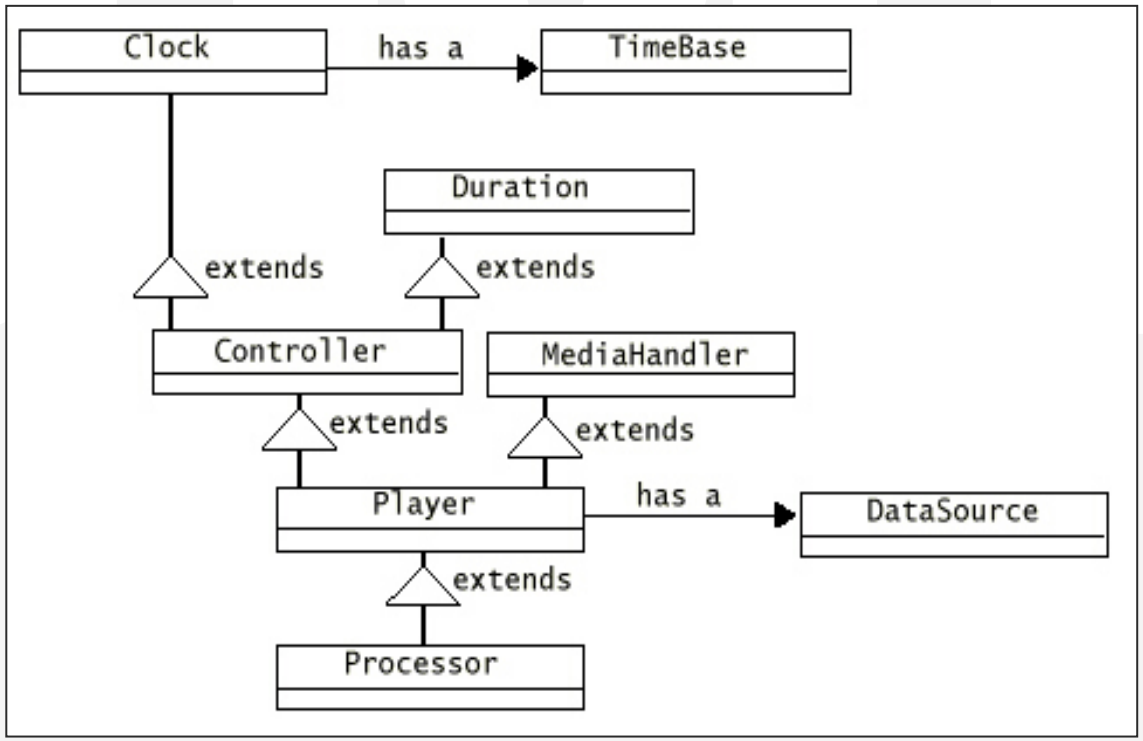
- Die Steuerung (Controls):
  - JMF-Control bietet einen Mechanismus für das Setzen und Abfragen von Objektattributen
  - Eine *Control* ermöglicht die Steuerung von Attributen über Benutzerschnittstellen
  - Einige Controls:
    - *CachingControl* (Steuerung des Downloadfortschritts)
    - *GainControl* (Regulierung der Lautstärke)
    - *StreamWriterControl* (Limitierung der Größe des erzeugten Stroms)

# Präsentation von Mediendaten (1)

- Präsentationsprozess wird modelliert durch das *Controller*-Interface:
  - *Controller* definiert die Zustände, die ein Medienkontrolleur (Objekt, das Medien steuert, präsentiert und erfasst) durchlaufen muss und bietet einen Mechanismus für die Steuerung der Übergänge zwischen den Phasen
  - *ControllerListener*-Interface: Muss von Klasse implementiert werden, die Events (Zustands-änderungen) vom Controller empfangen soll
  - 2 Medienkontrolleure (Interfaces):
    - *Player*
    - *Processor*

# Präsentation von Mediendaten (2)

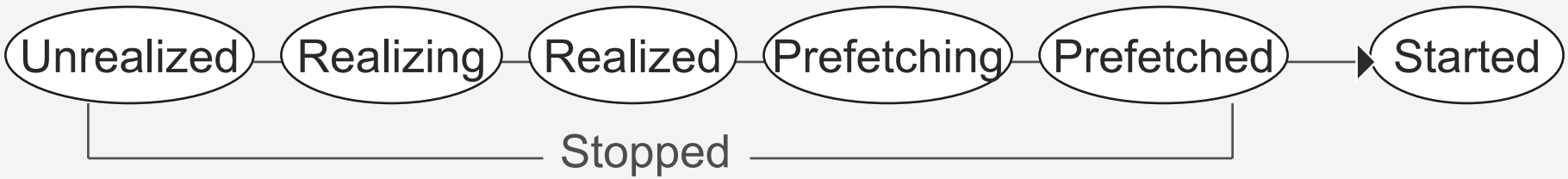
- JMF Controllers:




Ein *Player* oder *Processor* pro Datenquelle !

# Präsentation von Mediendaten (3)

- *Player*:
  - Ablauf: Datenquelle liefert Eingabestrom von Mediendaten an Player. Player verarbeitet Eingabestrom und gibt diesen wieder. Wiedergabegerät hängt von Mediendatentyp ab.
  - Player befindet sich in einem von 6 Zuständen:



# Präsentation von Mediendaten (4)

- *Player*-Zustände:
  - Wird Player erzeugt ist er im unrealized-Zustand.
    *Player player = Manager.createPlayer(dataSource);*
  - *player.realize()* – Zustand realizing: Betriebsmittel und Anforderungen werden bestimmt
  - Zustand realized: Infos sind ermittelt
  - *player.prefetch()* – Zustand prefetching: Mediendaten werden vorgeladen und Betriebsmittel zugewiesen
  - Zustand prefetched: bereit für Start
  - *player.start()* – Zustand started: TimeBase-Time und MediaTime sind gemapped, Clock läuft, Präsentation kann beginnen

# Präsentation von Mediendaten (5)

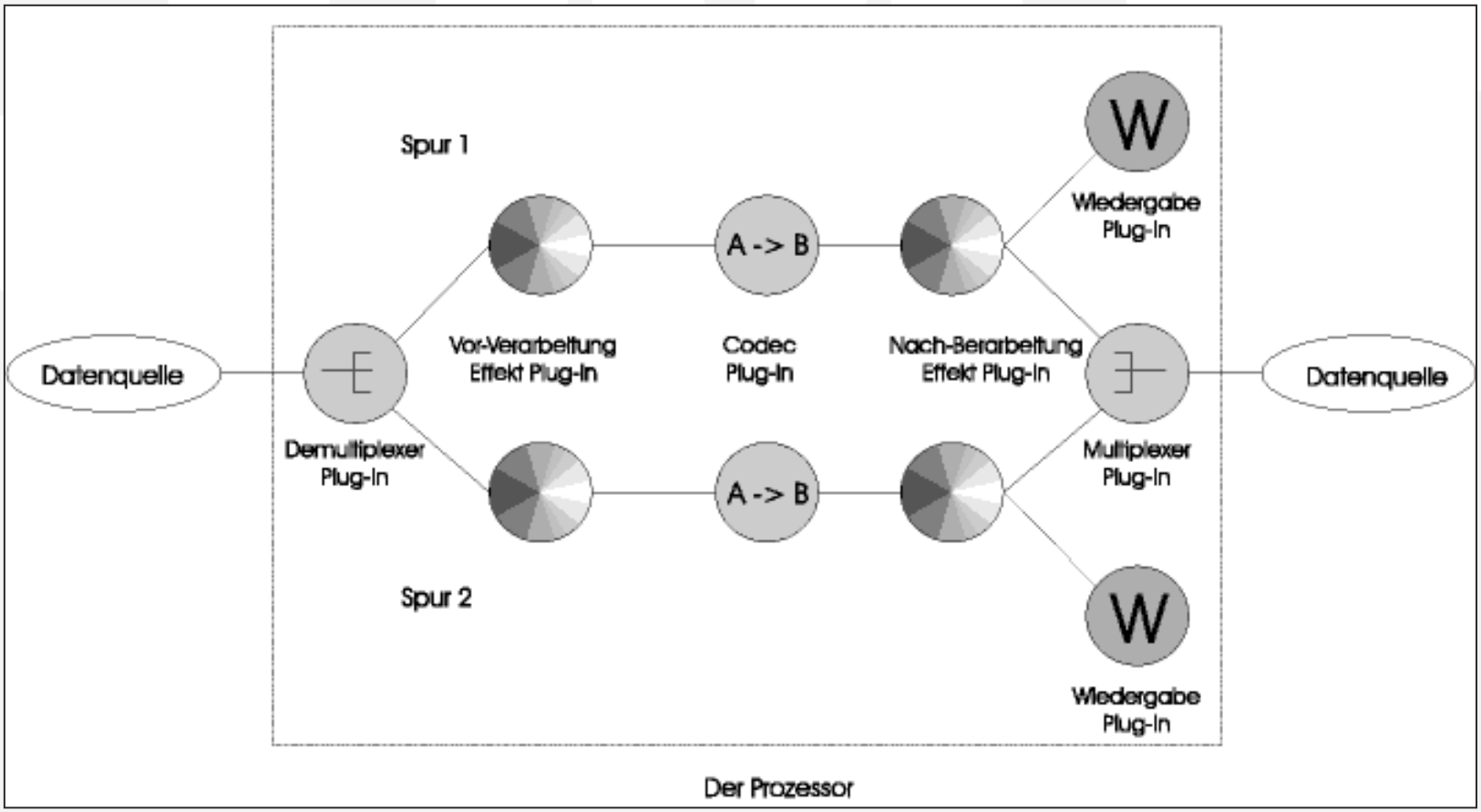
- Steuerung der Präsentation:
  - *getControls()*: liefert alle Controls eines Players oder Processors (z.B. GainControl)
- Benutzerschnittstelle:
  - *player.getControlPanelComponent();*
  - *player.getVisualComponent();*
- Events des Controllers:
  - Change (Attributänderungen z.B. *RateChangeEvent*)
  - Transition (Zustandsänderungen des Controllers)
  - Closed (ControllerClosedEvents)

# Verarbeitung von Mediendaten (1)

- *Processor*:
  - Ablauf: Datenquelle liefert wie bei *Player* Eingabestrom von Mediendaten an *Processor*. *Processor* verarbeitet Mediendaten und gibt diese an ein Präsentationsgerät (Playerfunktion) oder an eine neue Datenquelle aus. Diese Datenquelle dient dann einem anderen Player oder als Eingabe für einen *DataSink* (-> Datenspeicherung).
  - *Processor* erlaubt dem Entwickler, die Art der Verarbeitung zu definieren (↔ Player). Dadurch möglich: Anwendung von Effekten, Mischen, und Komponieren in Echtzeit.

# Verarbeitung von Mediendaten (2)

- Phasen der Verarbeitung:



# Verarbeitung von Mediendaten (3)

- Für jede Phase ein anderes JMF Plug-In:
  - *Demultiplexer* (Extraktion der Spuren)
  - *Effect*
  - *Codec*
  - *Multiplexer* (Merging mehrerer Spuren)
  - *Renderer* (Visualisierung/ Übertragung/ Ausgabe des Stroms)

# Verarbeitung von Mediendaten (4)

- Zusätzliche Zustände:
  Configuring & Configured:
  - Zwischen unrealized und realizing
  - Configuring: Processor verbindet sich mit Datenquelle, demultiplexed den Eingabestrom, und greift auf Infos über das Eingabeformat zu.
  - Configured: Processor ist mit Datenquelle verbunden und Datenformat ist festgelegt. *getTrackControls()* kann aufgerufen werden um für jede einzelne Spur eine *TrackControl* zu erhalten (Steuerung der Verarbeitung einer Spur)

# Verarbeitung von Mediendaten (5)

- Steuerung der Verarbeitung:
  - *TrackControl*: eine *TrackControl* pro Spur; per *getControls()* Zugriff auf Codec-Controls wie BitRate-, QualityControl
  - *setFormat(Format format)*: Formatspezifikation, z.B.

    *trackControl.setFormat(format);*
- Ausgabe der verarbeiteten Mediendaten:

  Die auszugebende Datenquelle wird wie folgt ermittelt:

  *DataSource output = processor.getDataOutput();*

# Erfassung von Mediendaten

- Engl: Capture
- Erfassungsgeräte (Capture Devices) können als Quelle für Multimediadaten dienen (z.B. Mikrofon erfasst Audiodaten)
  Vereinfacht: Datenquellen (DataSources)
- Beispiel: Gerät, das zeitgerechte Lieferung von Daten unterstützt, wird als *PushDataSource* gehandelt (z.B. Webcam)

# Daten-Speicherung und -Übertragung (1)

- *DataSink*:
  - Liest Mediendaten von einer Datenquelle und übergibt sie an ein Ziel (üblicherweise kein Präsentationsmedium !)
  - *DataSink* kann Daten in eine Datei schreiben, Daten über das Netzwerk schreiben, oder als RTP-Broadcaster funktionieren.
  - *StreamWriterControl*: zusätzliche Steuerung des Schreibens in Dateien (z.B. Einschränkung der Zieldateigröße)
  - Steuerung der Speicherung durch *DataSinkEvents* (z.B. *EndOfStreamEvent*) und Implementierung des *DataSinkListeners*

# Daten-Speicherung und -Übertragung (2)

- RTP:
  - Real-Time Übertragungsprotokoll
  - Medienstreams können in Echtzeit über das Netzwerk übertragen werden (z.B. Videokonferenz über das Internet)
  - RTP als Media-on-Demand Applikation
  - JMF bietet für RTP 3 Packages:
    - javax.media.rtp
    - javax.media.rtp.event
    - javax.media.rtp.rtcp
  - Hauptmechanismen: Empfangen, Senden von RTP-Streams über das Netzwerk

# Erweiterbarkeit des JMF

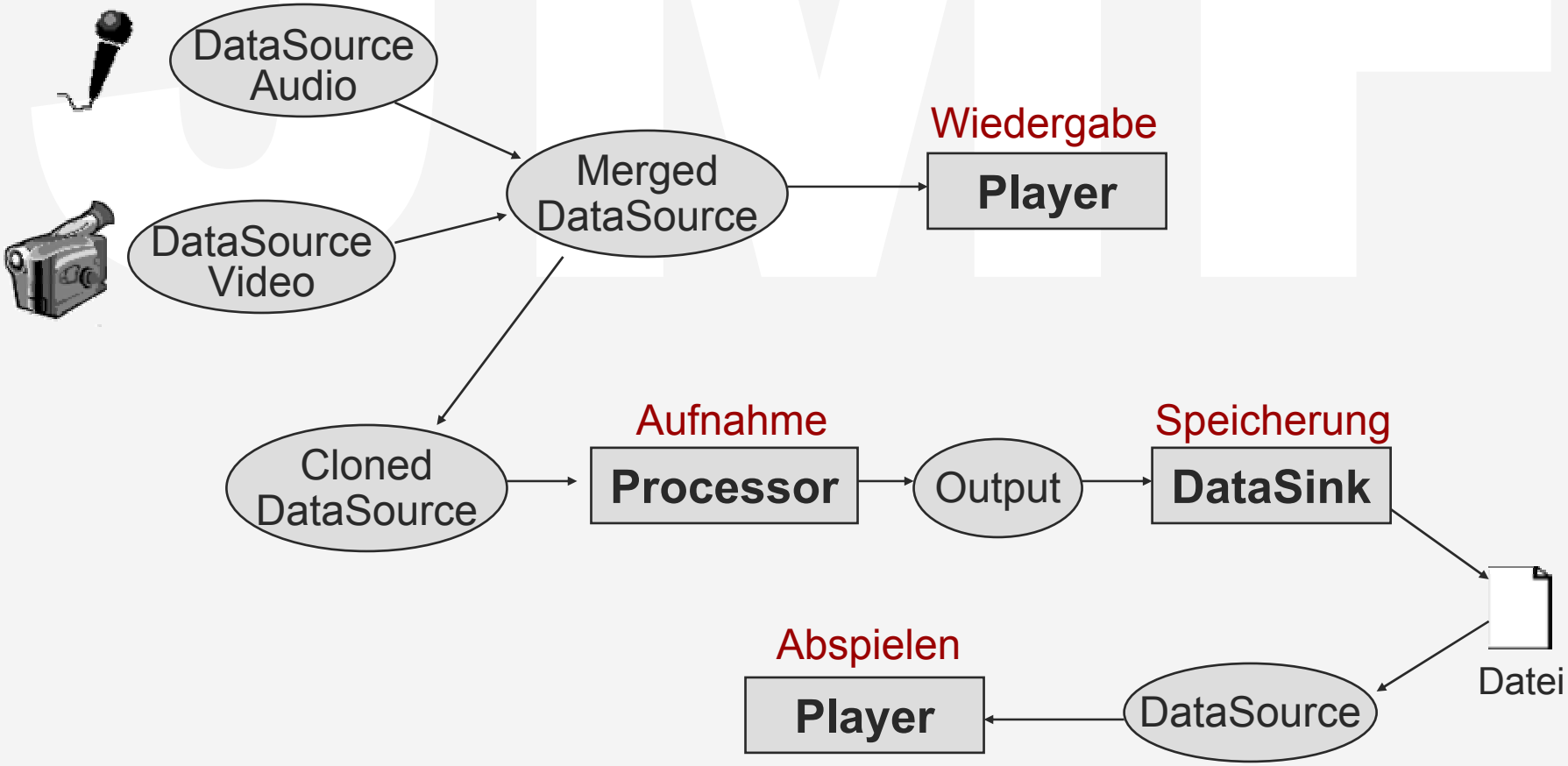
- JMF erweiterbar durch Implementierung von Plug-Ins, Media Handlern, und Datenquellen
- Plug-Ins: Erweiterung durch Implementierung eines der 5 Plug-In-Interfaces. Plug-Ins werden für den Processor verfügbar durch Registrierung mit dem *PlugInManager.* *PlugInManager.getPlugInList()* liefert Liste aller registrierten Plug-Ins.
- Mehr Flexibilität durch Implementierung von Media Handlern (*Player*, *Processor*, *DataSink*) und *DataSources*

# Aufgabe: JMF Terminal (1)

- Realisierung einer JMF-Anwendung zur synchronen Aufzeichnung und Wiedergabe von Audio und Video.
- Funktionen:
  - Durchgehende Wiedergabe von Audio (Mikro) und Video (Webcam)
  - Aufnahme von Audio und Video
  - Speicherung der Aufnahme in eine Datei
  - Abspielen der Aufnahme

# Aufgabe: JMF Terminal (2)

- Aufbau der Programmierung:



# Link: java.sun.com

*http://java.sun.com/products/java-media/jmf/*

- Installation (JMF 2.1.1)
- Dokumentation (JMF 2.1.1 API Guide)
- Spezifikation
- Beispiele
- Lösungen

# Weitere Links

- Nady Habashy: JMF zur Entwicklung multimedialer Anwendungen (deutsche Übersetzung des Kapitels „Understanding JMF des JMF Api Guides)

  *http://www.ubka.uni-karlsruhe.de/vvv/ira/2001/11/*

- Link-Seite:

  *http://www.informatik.uni-mannheim.de/informatik/pi4/stud/veranstaltungen/ws200102/mmprak/links.html*

# Merci für Ihre Aufmerksamkeit !

JMF