Benjamin Janzen, bj009

Daniel Knizia, dk100

Jannik Schlemmer, js329

Leonhard Lie, ll040

**Software Entwicklung 2: Software Projekt**

**Mediathek**

*GIT Repository: git@gitlab.mi.hdm-stuttgart.de:bj009/SE2\_Projekt\_Mediathek.git*

**Abstract**

Das Software Projekt „Mediathek“ ist eine Anwendung unter Linux zur Verwaltung und Wiedergabe von Video- und Audio-Dateien. Die Anwendung erlaubt das Einlesen von mp4, wma, ogg, mp3 und wav Dateien unter Angabe eines Ordnerpfades vom Benutzer. Die Dateien werden in einer gemeinsamen Liste gespeichert. Die Anwendung liest zudem beim initialen Start die Metadaten „Abspiellänge“, „Release Datum“, „Regisseur“ bzw. „Interpret“ und Genre selbstständig ein.

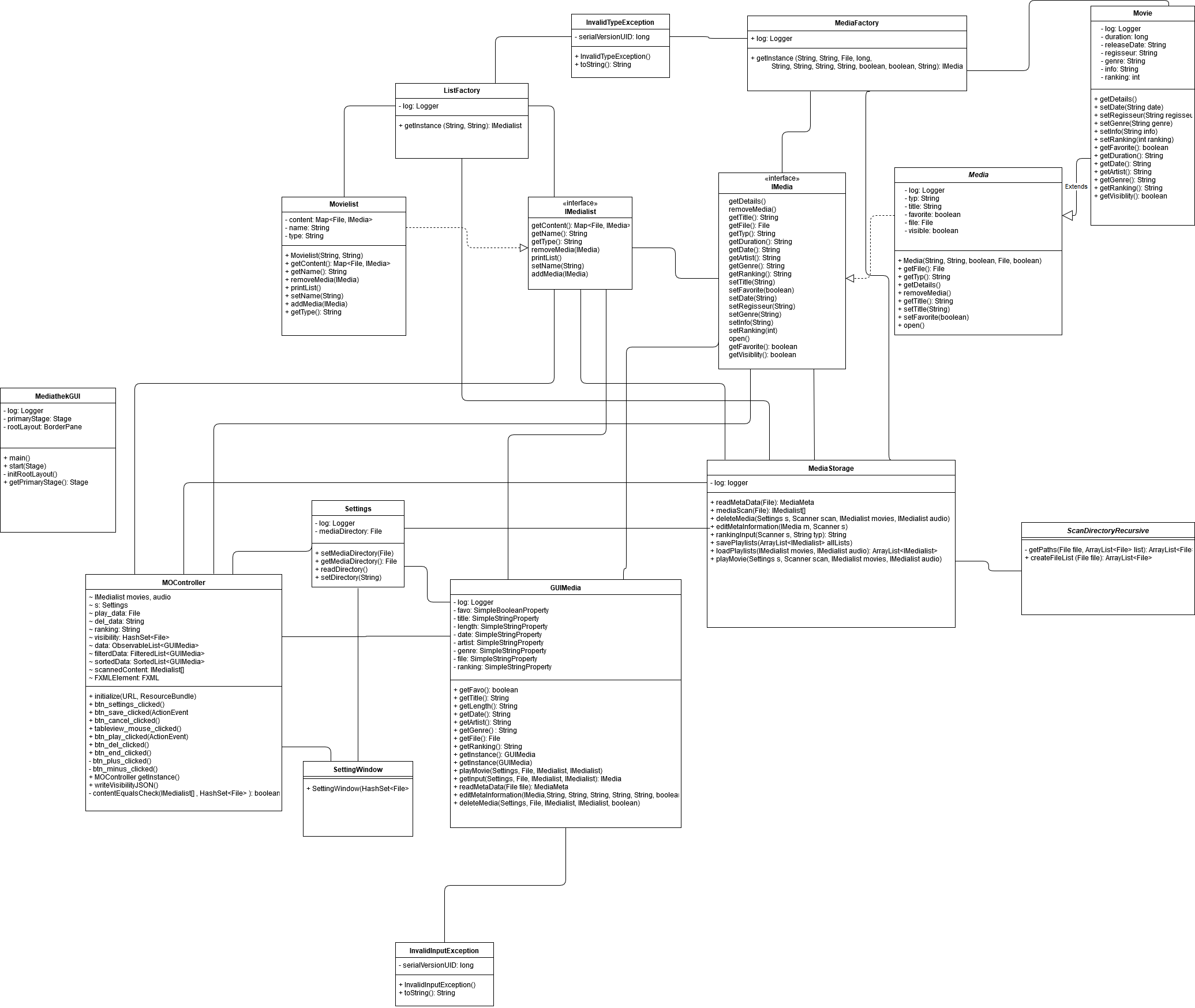
Dem Benutzer stehen nun zahlreiche Verwaltungsoptionen zur Verfügung:

* Anzeigen aller Mediendateien sowie deren Metadaten
* Editieren der Metadaten
* Gesondertes Anzeigen von Video- oder Audiodateien
* Abspielen der Mediendateien
* Playlistverwaltung:
  + Erstellen/Löschen/Speichern/Anzeigen/Umbennung von Playlists
  + Hinzufügen von Mediendateien
  + Löschen von Mediendateien (aus der Mediathek oder von der Festplatte)
* Änderung des Ordnerpfades
* Refreshfunktion durch Neueinlesen des Ordnerpfades

Die Anwendung lässt sich als Konsolenversion über die main Methode der App Klasse im Package de.hdm\_stuttgart.se2.softwareProject.mediathek.driver oder in Form eines GUI über die main Methode der MediathekGUI Klasse im Package de.hdm\_stuttgart.se2.softwareProject.mediathek.gui starten.

Die Konsolenversion bieten den vollen Funktionsumfang der vorangegangen Beschreibung und dient unter anderem der Funktionsüberprüfung und zum Debugging. Aufgrund der Inputabfrage des Benutzers gibt es in der Konsolenversion einige Systemausgaben über System.out.println(). Bei der GUI Anwendung ist die Playlistverwaltung noch nicht implementiert. Darüber hinaus steht in beiden Versionen noch eine Implementierung von Dokument-Typen wie PDFs in Aussicht.

**UML Klassendiagramm**



|  |  |  |  |
| --- | --- | --- | --- |
| **Funktion** | **Package/Klasse/Interface** | **Implementierung/Refe-renzierung** | **Kommentar** |
| Interfaces/  Vererbung | IMedia. IMedialist, Media | * Die abstrakte Klasse Media implementiert IMedia * Movie und Audio erben von Media * MovieList und AudioList implementieren IMedialist | Movie und Audio implementieren IMedia und erben von der abstrakten Klasse Media. Movielist und Audiolist implementieren IMedialist. Die Interfaces enthalten die gemeinsame Schnittmenge an Methoden, die für die unterschiedlichen Medientypen benötigt werden. Die abstrakte Klasse enthält die Schnittmenge der benötigen Felder. Referenzvariablen und Parameter sind nehmen jeweils den Interface Typen an , sodass die referenzierten Objekte flexibel implementiert werden können. |
| Package-Struktur[[1]](#footnote-1) | controller, driver, exceptions, gui, interfaces, lists, models | - | Packages sind nach dem Prinzip „Package by Layer“ angelegt worden. Die Packages gruppieren somit ähnliche Typen gem. der Definition des Oracle Tutorials für Packages[[2]](#footnote-2). |
| Exceptions | InvalidInputException, InvalidTypeException | ListFactory: getInstance()  MediaFactory: getInstance() | Eigene checled Exceptions zum Auffangen falscher Benutzereingaben. Auftreten der Exceptions auf Benutzerseite außerhalb des Einflussbereiches der Codeschreibenden, daher erben diese von java.lang.Exception. Auffangen der Exceptions mit anschließendem Logging der Exception (s. Logging Funktion). |
| Grafische Oberfläche (JavaFX) | - | MediathekGUI, MOController, SettingsWindow, filme.fxml, musik.fxml, MediathekOverview.xml main.css | Start() Methode in MediathekGUI: Initialisierung der Stage durch Übergabe der Scene aus der MediathekOverview.fxml.  MOController als Controller Klasse für das Event Handling der MediathekOverview.fxml. SettingWindow für das Settings Fenster inkl. Event Handling.  Verschachteltes Layout: BorderPane als Root Node. Verschachtelte Container (VBox. HBox, TableView etc.) und Controller (Button, RadioButton, TextField, Label etc.) Elemente. Controller Elemente mit Event Handling.  GUIMedia mit Methoden zum Verwalten der Listen und der Mediendateien. |
| Logging | log4j2.xml |  | * Logger für jedes Package (getLogger() Methoden mit Klassennamen als Parameter loggen automatisch in den jeweiligen Package Logger) * Log Level:   + Error: bei kritische Fehlern, die Potentiell die Anwendung beenden (exceptions + root Logger)   + Info: bei Benutzerinteraktionen (gui + controller Logger)   + Debug: Bei fehleranfälligen Methoden/Klassen (driver, models, lists Logger) * .log Dateien füt jeden Package Logger + Root Logger * Logger für die gesamte Logging-Historie (append=“true“, ansonsten aus Gründen der wirtschaftlichkeit „false“) |
| UML | - | - | Siehe UML Diagramm S. 3. |
| Threads | - | MOController:   * public void initialize() | Threads zum Initialen bzw. Wiederholten Einlesen des Inhaltes des Ordnerpfades in eine Movie und Audio Instanz zum Updaten der TableView parallel zum main Thread. Wiederholen des Einleseprozesses **jede Sekunde**. Synchronisierung kritischer Abschnitte durch lock() und unlock(). Kritische Abschnitte schreiben in gemeinsam nutzbare HashSet Objekte, die zur Darstellung der Liste von Bedeutung sind. |
| Streams u. Lambda-Funktionen | - | GUIMedia:   * public static IMedia getInput() | Verwendung eines parallel Streams mit ArrayList als Datenquelle zum Filtern von IMedia Objekten entsprechend einer File Objekt Referenz. Verwendung von .filter() und .findAny() als intermediate Operation und .get() zur Rückgabe des gesuchten IMedia Objektes und anschließendem Terminieren des parallelen Streams.  Verwendung eines Lamda Ausdruckes für eine konkrete Predicate Implementierung als Übergabeparameter an die .filter() Methode. |
| Dokumen-tation und Test-Fälle | - | - | Tests der MediaFactory und der MedialistFactory:   * Positivtest: Test auf Richtigkeit der jeweiligen return Typen bei Übergabe der korrekten Typenargumente * Negativtest: Test auf werfen der InvalidTypeException bei Übergabe eines ungültigen Typen   Test hinsichtlich der Änderung der Mediendateien:   * Positivtest: Testen des korrekt durchgeführten Löschens einer Datei von der Festplatte. * Negativtest: Test auf werfen der InvalidInputException bei Übergabe eines ungültigen Typen |
| Factories | MediaFactory, ListFactory | MediaStorage:   * loadPlaylists() * mediaScan() | Movie, Audio, Book und Movielist, Audiolist, Booklist Objekte, werden in der Anwendung jeweils durch Factories initialisiert. Factories akzeptieren Medientypen Video, Audio und Book als Argumente und geben entsprechen den Typen Movie, Book, Audio (MediaFactory) oder Movielist, Booklist, Audiolist (MedialistFactory) zurück.  **„Korrekte Sichtbarkeit der implementierenden Klassen“?** |

1. Die aufgeführten Pakete gehören zum Namensraum de.hdm\_stuttgart.se2.softwareProject [↑](#footnote-ref-1)
2. https://docs.oracle.com/javase/tutorial/java/package/packages.html [↑](#footnote-ref-2)