Relazione per "Programmazione ad Oggetti"

Alex Mazzoni
Samuele Casadei
Christian Luca
Matteo Zoli

16 febbraio 2024

Indice

Capitolo 1

Analisi

ElektReader è un software dedicato a riproduzione e gestione locale di file in formato audio, esso si occuperà di fornire le funzioni base di riproduzione e navigazione di tracce musicali tramite playlist. In quanto locale, l'ambiente di lavoro sarà deciso dall'utente, e il software si occuperà in autonomia di astrarre una gerarchia di playlist e brani dall'ambiente fornito.

1.1 Requisiti

Requisiti funzionali

- Il software dovrà occuparsi di appiattire la gerarchia dell'ambiente, fornendo le sole playlist all'utente in modo immediatamente accessibile.
- I file corrotti, non leggibili o non supportati devono essere esclusi durante il caricamento dell'ambiente, lo stesso vale per le playlist considerate vuote.
- Deve essere lasciata all'utente la possibilità di modificare l'ordine di riproduzione dei brani di una playlist.
- Devono essere forniti i comandi di manipolazione della riproduzione dei brani in un'apposita interfaccia grafica (come play, pause, loop, shuffle, ecc.).
- Possibilità di creare varianti modificate dei brani, tagliandone il contenuto.
- Possibilità di scegliere tra due diversi tipi esposizione grafica dei brani.

Requisiti non funzionali

- Interfaccia grafica user friendly con comandi semplici.

1.2 Analisi e modello del dominio

Alla base di un riproduttore MP3 vi è certamente il concetto di Song, che andrà astratto in funzione delle nostre necessità; esse andranno conseguentemente accorpate in Playlists. Uno degli aspetti più impegnativi sarà necessariamente riuscire ad astrarre e manipolare nel modo più semplice ma allo stesso tempo efficace possibile delle canzoni; difatti, esse in principio non saranno altro che un susseguirsi di bytes: sarà dunque necessario lavorarci in maniera adeguata. Per le playlist, che saranno semplicemente delle collezioni di Songs, sarà necessario individuare la struttura dati più adeguata allo scopo. Un altro concetto essenziale con cui dovremo avere a che fare è il modo di riprodurre le canzoni: bisognerà valutare il da farsi, in questo la javadoc e/o librerie esterne potranno certamente fornire delle possibilità per adempiere allo scopo.

Capitolo 2

Design

2.1 Architettura

Il pattern architetturale utilizzato è il classico MVC (Model-View-Controller); andando per gradi:
Model: Le classi principali utilizzate ai fini della risoluzione del problema, implementazioni delle interfacce contenute in Api:

- Reader: questa è la logica principale che suddivide gerarchicamente in più sottoproblemi, essa infatti racchiuderà la logica del lettore in sé, quindi tutta la logica in gradi di riprodurre e gestire manipolare caricare i file audio, si interfaccia con il filesystem, per caricare l'ambiente (una vera e propria radice dalla cartella inserita come ambiente la logica creerà le varie playlist (ovvero delle directory che contengono file audio NON CORROTTI supportati e validi) esse verranno poi intese appunto come playlist contenitori di brani, da questa classe sarà possibile manipolare tutte le sottoclassi responsabili della logica del lettore.

- PlayList: rappresenta l'astrazione di una raccolta di brani, condivide parte della logica filesystem, in quanto deve essere utilizzato dal reader durante la mappatura dell'ambiente; stabilisce un ordine implicito dei brani tramite un'indicizzazione fisica nei nomi dei file.

- Songs: rappresenta l'astrazione di un brano su cui mappare i file riproducibili all'interno dell'ambiente, contiene le informazioni necessarie alla riproduzione del brano, e alla sua visualizzazione grafica.

- MediaControl: Rappresenta il concetto di riproduttore di brani all'interno del progetto; esso avrà dunque il compito di riprodurre, selezionare determinate canzoni scelte dall'utente o proseguendo / regredendo

di una posizione, modificare volume etc. Tutto dovrà risultare quanto più user-friendly possibile all'atto pratico.

- TrackTrimmer: Rappresenta l'oggetto adibito al taglio di file multimediali (specificatamente file mp3 e wav). Esso quindi offre all'utente: la possibilità di selezionare il file da tagliare dal file system, non limitando così la scelta ai soli brani in uso nel riproduttore, e successivamente, dopo aver inserito i parametri necessari, creare il nuovo file tagliato.

View: Le classi che compongono la componente adibita alla sezione grafica del progetto sono le seguenti:

- GUI: Ricordando che la componente grafica del progetto è stata sviluppata mediante la libreria esterna JavaFX, Questa classe avrà il semplice scopo di inizializzare l'ambiente principale, settando determinate dimensioni, varie proprietà e caricando il file .FMXL necessario al nostro scopo.

- TrimGUI: Questa classe ha il compito di generare un nuovo stage per effettuare il taglio dei brani.

- App: Pur non essendo inserito nel package view, è doveroso menzionare questa classe poiché è colei che contiene il metodo main necessario a lanciare l'applicazione.

Controller: Queste classi hanno lo scopo di scorporare la parte grafica in sottoparti, manipolando queste ultime attraverso i metodi implementati nelle omonime classi. Elencandole, troviamo:

- GUIController: Questo è il controller principale quello che include lo scheletro dell'applicazione, ovvero il controller che gestisce il file FXML principale, poi da essa verranno poi costruiti i vari controller per suddividere in sottoparti dinamiche la gui.

- MediaControlsController: è il controller che si occupa di gestire, riempendo delle informazioni necessarie quando di bisogno il pannello relativo al riproduttore musicale.

- PlayListsController: è il controller che si occupa della visualizzazione delle playlist e della selezione grafica di esse da parte dell'utente.

- SongsController: Ogni qual volta viene aperta una playlist mostra all'utente le canzoni contenute con la possibilità di selezionarle e riprodurle.

- TrackTrimmerController: Controller che gestisce lo scambio di informazioni tra TrimGUI (generata dalla creazione del controller) e TrackTrimmer.

2.2 Design dettagliato

ReaderImpl: Classe implementativa che si occupa di implementare la logica fondamentale del lettore, essa è costituita in modo gerarchico: incominciando da un environment ovvero da una radice la logica, setCurrentEnvironment -¿ inizializza l'ambiente principale del progetto, esso infatti crea le playlist, passandogli solo quelle volute dalle classi statiche della classe Reader: "getAndFilterSongs" e "isSupportedSong", queste funzioni sono in grado di filtrare tutti i file, le playlist infatti vengono create in base ai seguenti filtri: una playlist é considerata tale se contiene almeno una canzone, una canzone è considerata supportata se è di una dimensione minima (100KB) e é un tipo di file che l'applicazione é in grado di gestire, senza creare direttamente le canzoni per risparmiare risorse e tempo. Non sono gestite le canzoni duplicate (con lo stesso nome). Avendo impostato la classe in maniera gerarchica appena l'environment viene impostato correttamente, anche le altre funzioni saranno rese disponibili, e verrà anche istanziato un oggetto MediaControls, responsabile per la riproduzione delle canzoni. la classe quindi presenta tutte le playlist presenti nell'environment, per questo sono presenti tutti i metodi per visualizzarle come: restituire la lista di tutte le playlist, prendere una playlist dato un percorso, oppure restituire il numero totale di playlist.

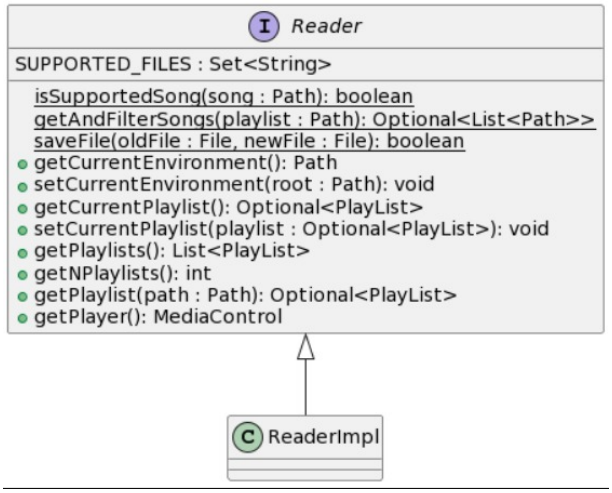

Figura 2.1: UML illustrativo della classe ReaderImpl con relativa interfaccia.

GUIController: Questa classe è responsabile della creazione del controller principale dell'applicazione principale, essa infatti viene usata come Controller dall'app.FXML, è strutturata nella seguente maniera: carica al suo interno tutti gli elementi contenuti nell'FXML, e inizializza poi degli elementi fondamentali che garantiscono dinamicità e sincronicità durante l'esecuzione: infatti per rendere l'applicazione dinamica e sincronizzata vengono utilizzati dei pannelli specifici inizialmente vuoti che andranno ad essere reinizializzati e riempiti in base alle scelte dell'utente, essi sono appunto gli altri controller: MediaControlController -¿ responsabile per il pannello dei comandi di riproduzione dei brani. SongController -¿ responsabile per il pannello delle canzoni. PlaylistController -¿ responsabile per il pannello delle playlist.

essi vengono poi sempre richiamati e riaggiornati, da un Thread sempre in ascolto pronto a mantenere attivi ed aggiornati tutti gli elementi della GUI con la logica del programma, contenuta in READER, campo statico che si occupa di gestire la logica della GUI.

initialize è un metodo ereditato dalla classe implementata Application, è utilizzato per inizializzare uno stato base appena viene caricato il file FXML nella classe, viene utilizzato ad esempio per caricare automaticamente un environment.

FindController: è un pannello che può comparire con la pressione del tasto Find, esso infatti viene creato al momento, e mette a disposizione un textField, utile per cercare le canzoni e le playlist, viene creato in modo swing, esso poi rimane in ascolto ogni volta che l'utente digita un carattere sulla textField, cercando poi tra canzoni e playlist, in base a diversi pattern: - durata (hh:mm:ss) -¿ ricerca le canzoni con la durata e le playlist con la durata specificata. - indici (numero) -¿ ricerca le canzoni con l'indice scelto e non cerca nessuna playlist - stringa -¿ se non corrisponde a uno dei due pattern avviene la ricerca per stringa non completa (è importante il camelCase). per cercare e aggiungere le varie query ho utilizzato due predicate, che sono andato poi a modificarmi in base al pattern che ho scelto, in questo modo evito qualsiasi ripetizione del codice, e sfrutto gli stream, ogni risultato della ricerca poi potrà essere cliccato, per essere riprodotto, grazie al Thread sempre in esecuzione contenuto in GUIController, la GUI verrà sempre aggiornata anche se non segue un ordine di eventi come in questo caso.

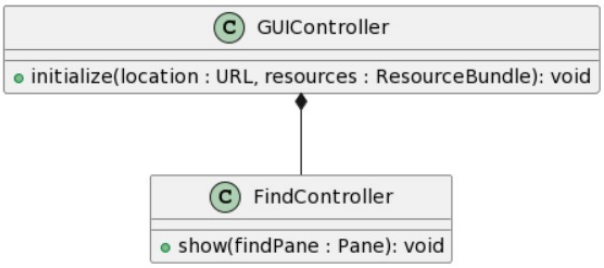

Figura 2.2: UML illustrativo della classe GUIController con relativa interfaccia.

QueueGUI: questa classe contenuta in view é responsabile della creazione di un nuovo stage che contiene semplicemente la coda di riproduzione, è creato in maniera swing, e non possiede particolari aggiunte.

Mp3MediaControl: Classe implementativa che si occupa della gestione della riproduzione delle Songs incorporate in Playlists. Facendo ampio uso di Optional, si riesce inoltre ad evitare tutte le principali eccezioni che potrebbero scaturire a run-time.
Alla base di tutto vi è il MediaPlayer: esso è un oggetto di riproduzione file multimediali appartenente alla libreria di terze parti JavaFX, e che dispone al suo intento di numerosi metodi utili al nostro scopo. Molti metodi, come play, pause e stop risulteranno essere dunque dei semplici alias di questi ultimi, con l'accortezza di verificare che i campi opzionali a cui devono accedere siano effettivamente riempiti. Al fine di ottenere le canzoni da riprodurre, ho istanziato un campo Optional¡List¡Song¿¿ playlist, che andrà settato da parte del metodo setPlaylist. Internamente la soluzione utilizzerà un indice interno intero che permetterà in modo rapido e semplice di impostare la canzone successiva, precedente oppure una scelta a discrezione dell'utente. Alcuni metodi saranno solamente utilizzati dalla controparte controller MediaControlsController ai fini di view, come ad esempio getNextSong, getCurrentSong e getStatus. I metodi loop e rand si occupano rispettivamente di gestire i stati dell'opzione loop (disabilitata, sulla playlist e sul brano) attraverso la modifica del comportamento del metodo nextSong e randomizzare (o ripristinare allo stato originario) la playlist.

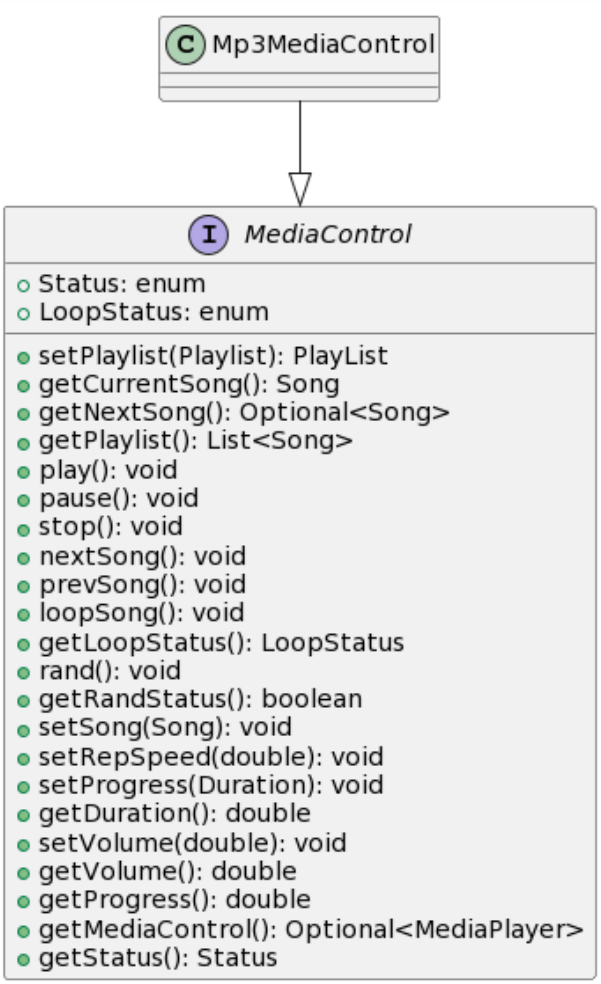

Figura 2.3: UML illustrativo della classe Mp3MediaControl con relativa interfaccia.

MediaControlsController: classe utilizzata da GUIController che ha come obiettivo quello di far ottenere alla componente grafica del progetto componenti, event Handler, listeners, risorse di carattere multimediale quali immagini in formato png e quant'altro. A costruttore ottiene un pannello container vuoto e uno Slider che fungerà da progressBar della canzone attualmente riprodotta. Essa contiene un metodo molto importante che si chiama reload(): esso è essenziale al fine di ricaricare tutte le componenti che devono variare di aspetto e/o funzione al momento della selezione di un altro brando da parte dell'utente o del sistema.

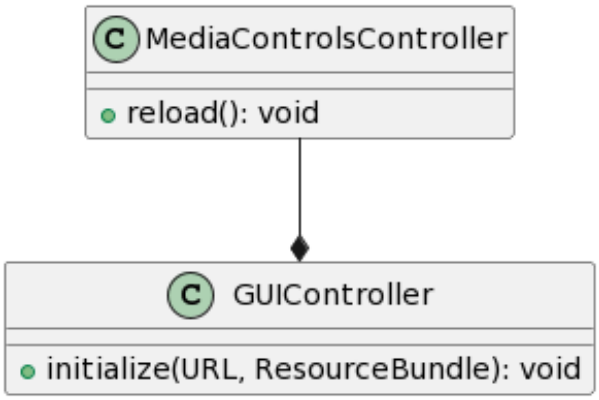

Figura 2.4: UML illustrativo della classe Mp3MediaControl.

Mp3Song: Classe implementativa di Song, si occupa di fornire a MediaPlayer una rappresentazione di brano semplice e completa, mettendo a disposizione tutte le informazioni sul brano.
Essenzialmente un brano consiste in un file formato audio, dunque lo scheletro di questa implementazione è un java.io.File, essendo una classe utilizzata dal Reader, l'istanziazione di questa classe è completamente orientata al filesystem, in quanto l'unico argomento di creazione è il percorso del file. Tutti i metodi di questa classe servono per reperire informazioni sul brano; a questo scopo viene utilizzata una libreria esterna JAudioTagger, che fornisce la classe AudioFile, rappresentante un file riproducibile; ogni AudioFile rende disponibile un Tag, ovvero un contenitore dei metadata del file strutturato come una mappa navigabile per campi detti "chiave". Questi campi permettono di reperire informazioni come artista, album e genere (se presenti nei metadata); per ragioni implementative si è scelto di utilizzare il nome del file come titolo effettivo, mentre AudioFile fornisce i metodi di base per estrarre la durata del brano.

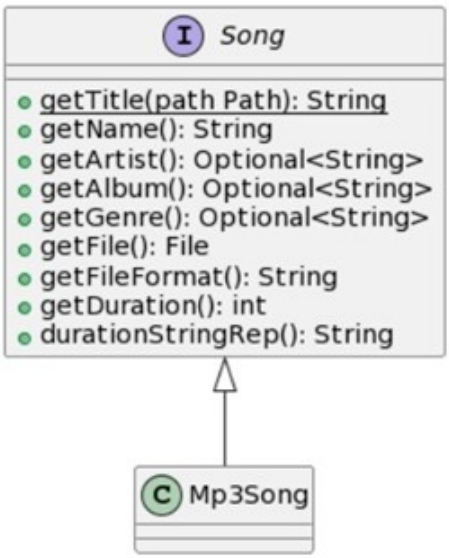

Figura 2.5: UML illustrativo della classe Mp3Song.

Mp3PlayList: Classe implementativa dell'interfaccia Playlist offre metodi per la ricerca di brani, e manipolazione di questi ultimi. È possibile ricercare una canzone per riferimento, indice o percorso; fornisce una Lista immutabile di canzoni tramite il metodo getSongs(). La soluzione all'ordinamento dei brani è stata modificare il nome di ogni file applicando un indice numerico al momento dell'istanziazione: in questo modo ogni nome di file avrà la struttura "indice – nome file". Non è necessario che ogni file venga indicizzato, dei file verranno correttamente rispristinati ad ogni caricamento dell'ambiente. È da specificare che in caso di parità tra indici l'esito dell'ordinamento è incerto, ogni altro tipo di conflitto è gestito.

(es. [1, 13, 4, 20, 18] -> sorting -> [1 ,4, 13, 18, 20] -> saving as -> [1, 2, 3, 4, 5]
[-1, 10, 2 ,10] -> sorting -> [-1, 2, ? ,?] -> saving as [1, 2, 3, 4])

Mp3Playlist fornisce anche un metodo statico per estrarre l'indice a partire dal nome del file, in quanto risulta utile anche al Reader per eventuali rimappature del file system.

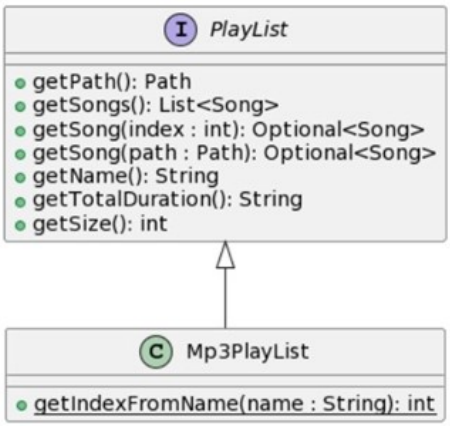

Figura 2.6: UML illustrativo della classe Mp3Playlist.

PlayListsController: Questa classe viene utilizzata da GUIController per mostrare graficamente tutte le playlist presenti nell'ambiente, la sua istanziazione richiede un pannello che di fatto sarà la view della classe playlist, mentre il model (ovvero l'ambiente corrente) viene ottenuto dal campo statico READER di GUIController. Ogni volta che un ambiente viene caricato, viene richiamato il metodo reload(), che riempie il pannello con i bottoni delle playlist presenti (creati tramite un metodo privato), ad ogni modifica della finestra, viene chiamato responsive(), che effettua una resizing delle componenti grafiche di PlayListsController. Questa classe fa anche uso di un SongsController, dato che le canzoni mostrate corrispondono solamente a quelle della playlist corrente, l'implementazione di due view differenti per le canzoni è stata gestita con un flag interno di PlayListsController tramite il quale si decide di richiamare uno specifico metodo load() del SongsController().

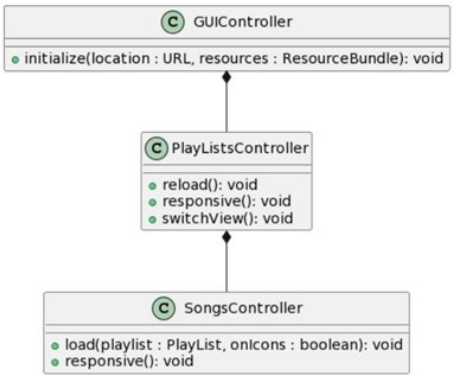

Figura 2.7: UML rappresentativo di PlayListsController e SongsController.

SongsController: Classe utilizzata da PlayListsController per istanziare le anteprime delle canzoni nella view, ogni qual volta viene cambiata o scelta una playlist, PlayListsController richiama il metodo load(Playlist, boolean) di SongsController, dove playlist è la nuova playlist da cui estrarre le canzoni (le anteprime delle canzoni vengono create da un metodo privato interno) e un flag che indica che tipo di view caricare; questa classe dispone di due tipi di view intercambiabili, lo scambio avviene tramite il pulsante "view" dell'interfaccia grafica. Il primo tipi di view è rappresentata da icone contenti titolo e durata della canzone, se cliccate settano il brano corrente e ne avviano la riproduzione. Il secondo tipo di view sostituisce le icone con una lista utile per avere qualche informazione in più sui brani, l'interazione con le voci di essa ha lo stesso effetto del cliccare sulle icone del precedente metodo di visualizzazione.

TrackTrimmerController: Classe adibita alla gestione dello scambio di informazioni tra TrackTrimmer e TrimGUI .

TrackTrimmerImpl: La classe implementa l'interfaccia TrackTrimmer, offrendo la logica per selezionare la traccia da tagliare e per creare la nuova traccia secondo i parametri forniti dall'utente. Il problema che è sorto immediatamente era riguardante la parte di taglio, poiché, dopo una breve ricerca, ho scoperto che la codifica di un file audio dipende da più fattori del previsto. Di conseguenza ho optato, dopo aver cercato online delle librerie per la manipolazione di file multimediali, all'uso della libreria JAVE 2 (Java Audio Video Encoder) che, in maniera abbastanza intuitiva, mette a disposizione, tra le altre, le classi MultimediaObject (dato un File multimediale crea un oggetto che lo rappresenta), AudioInfo (un oggetto di tale classe viene utilizzato per estrarre i dati riguardanti la codifica del file originale), AudioAttributes (un oggetto di questa classe mantiene i dati precedentemente estratti, per poi passarli all'oggetto EncodingAttributes), EncodingAttributes (il cui oggetto mantiene i dati passati da AudioAttributes e il formato del file che verrà generato) ed infine Encoder (l'oggetto Encoder viene utilizzato per effettuare l'effettivo encoding del nuovo brano, dato il MultimediaObject del file orginale, il file target e gli EncodingAttributes).

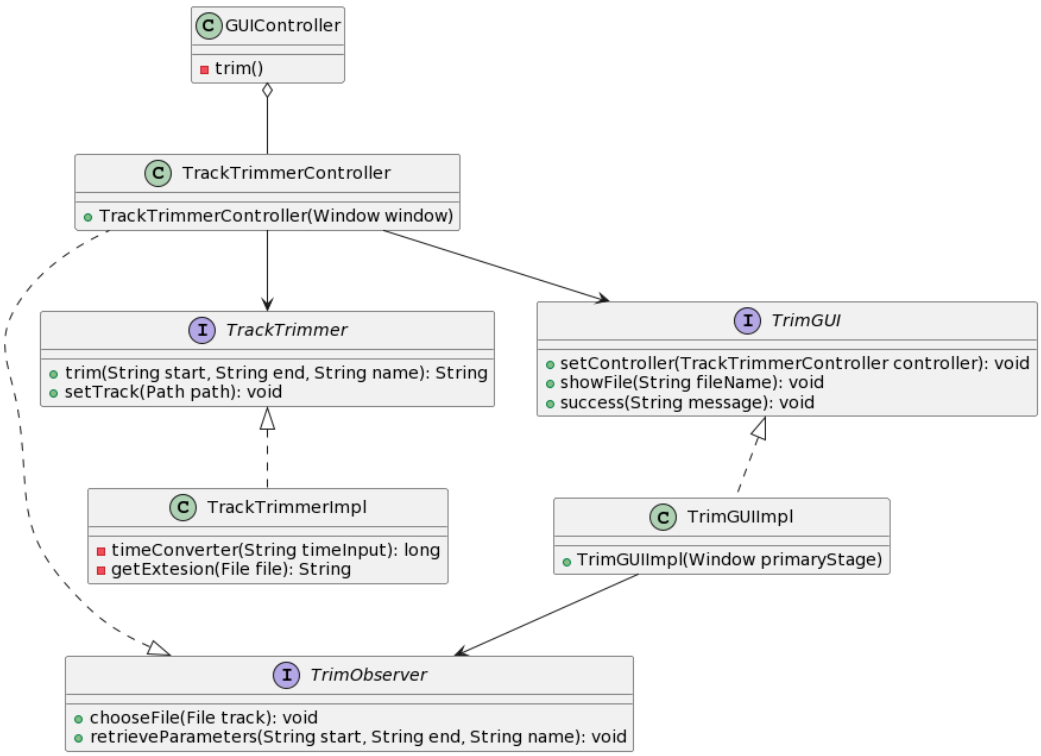

Figura 2.8: UML rappresentativo dell'applicazione di MVC tra TrackTrimmer, TrimGUI e TrimObserver.

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Ai fini di testing automatizzato ci siamo affidati a JUnit Platform. Ogni componente chiave del progetto ha un metodo di testing che permette di verificarne le principali funzionalità. Vengono utilizzati dei campi costanti che verificano che le operazioni eseguite dal nostro software siano effettivamente funzionanti.

3.2 Note di sviluppo

Alex Mazzoni:

- Uso della libreria esterna javaFX, con utilizzo di FXML e quindi anche SceneBuilder.
  `https://github.com/pallax03/OOP23-ElektReader/blob/41ec1a1a4d201d9c192e6e16c`
  `app/src/main/java/elektreader/view/GUI.java#L41`

- suddivisione dei vari pannelli dinamici della GUI.
  `https://github.com/pallax03/OOP23-ElektReader/blob/41ec1a1a4d201d9c192e6e16c`
  `app/src/main/java/elektreader/controller/GUIController.java#`
  `L301`

- Uso di stream, lambda expression e predicate per la ricerca di canzoni e playlist, li ho usati anche in altri classi.
  `https://github.com/pallax03/OOP23-ElektReader/blob/41ec1a1a4d201d9c192e6e16c`
  `app/src/main/java/elektreader/controller/FindController.java#`
  `L101`

- Utilizzo di Runnable + javafx listener per sincronizzare la logica con la gui.
  https://github.com/pallax03/OOP23-ElektReader/blob/41ec1a1a4d201d9c192e6e16c
  app/src/main/java/elektreader/controller/GUIController.java#
  L271

- Utilizzo di regex per trovare il pattern selezionato.
  https://github.com/pallax03/OOP23-ElektReader/blob/41ec1a1a4d201d9c192e6e16c
  app/src/main/java/elektreader/controller/FindController.java#
  L82

- implementazione per prendere solo l'environment (playlist, song) ritenuto opportuno, in questo modo le altri classi si aspettano sempre dei percorsi validi.
  https://github.com/pallax03/OOP23-ElektReader/blob/41ec1a1a4d201d9c192e6e16c
  app/src/main/java/elektreader/model/ReaderImpl.java#L35

- Ampio utilizzo di opzionali, qui un riferimento, ma si trovano in molti metodi della classi.
  https://github.com/pallax03/OOP23-ElektReader/blob/41ec1a1a4d201d9c192e6e16c
  app/src/main/java/elektreader/api/Reader.java#L110

Samuele Casadei:

- Uso della libreria esterna jaudiotagger per la lettura di metadati da file
  https://github.com/pallax03/ElektReader/blob/3e07095f7f69d18ea34da76e30ca9aa
  app/src/main/java/elektreader/model/Mp3Song.java#L61

- E anche per l'estrazione della durata dei brani.
  https://github.com/pallax03/ElektReader/blob/4968f11e074e5eeed540e68056afce0
  app/src/main/java/elektreader/model/Mp3Song.java#L71

- Uso di stream e lambda expression per il sorting per indice dei brani delle playlist.
  https://github.com/pallax03/ElektReader/blob/4968f11e074e5eeed540e68056afce0
  app/src/main/java/elektreader/model/Mp3PlayList.java#L135-L138

- Utilizzo di Tooltip (javafx.scene.control) per mostrare le informazioni dei brani.
  https://github.com/pallax03/ElektReader/blob/4968f11e074e5eeed540e68056afce0
  app/src/main/java/elektreader/controller/SongsController.java#
  L61-L63

- Utilizzo di regex per la manipolazione delle stringhe.
  https://github.com/pallax03/ElektReader/blob/4968f11e074e5eeed540e68056afce0
  app/src/main/java/elektreader/model/Mp3PlayList.java#L186-L194

- Ampio utilizzo di opzionali, qui un riferimento, ma si trovano in molti metodi della classi.
  https://github.com/pallax03/ElektReader/blob/4968f11e074e5eeed540e68056afce0
  app/src/main/java/elektreader/model/Mp3Song.java#L60

Christian Luca:

- Uso di lambda expression con stream per verificare che la canzone che viene passata come parametro sia effettivamente facente parte della playlist attualmente settata.
  https://github.com/pallax03/ElektReader/blob/9f262256f459d369bed5c9038a5b480
  app/src/main/java/elektreader/model/Mp3MediaControl.java#L239

- Setting di un mediaPlayer lavorando con opzionali, passando come parametro un Media che però dovrà essere lavorato poiché c'è bisogno di una catena di conversione da Song fino a Media.
  https://github.com/pallax03/ElektReader/blob/9f262256f459d369bed5c9038a5b480
  app/src/main/java/elektreader/model/Mp3MediaControl.java#L77

- Controllo dell'effettiva presenza di un campo settato come opzionale per prevenire NoSuchElementException a run-time. Quasi tutta la classe del permalink si basa su questi controlli, pertanto ne citerò uno soltanto.
  https://github.com/pallax03/ElektReader/blob/9f262256f459d369bed5c9038a5b480
  app/src/main/java/elektreader/model/Mp3MediaControl.java#L238

- ActionListener usato per settare il valore attuale della progressBar uguale alla durata attuale del mediaPlayer.
  https://github.com/pallax03/ElektReader/blob/e4785b931b6d7a32f08f33c03d98bb1
  app/src/main/java/elektreader/controller/MediaControlsController.
  java#L136

Matteo Zoli:

- Utilizzo della libreria JAVE per manipolazione dei file multimediali.
  https://github.com/pallax03/ElektReader/blob/f253e4b3457a58e4b7d5b25f6e8c255
  app/src/main/java/elektreader/model/TrackTrimmerImpl.java#L46

- Utilizzo di Regex per eseguire split di stringhe (anche nel metodo dopo).
  `https://github.com/pallax03/ElektReader/blob/f253e4b3457a58e4b7d5b25f6e8c255`
  `app/src/main/java/elektreader/model/TrackTrimmerImpl.java#L78`
- Dichiarazione di nuovo stage e suoi componenti con JavaFX.
  `https://github.com/pallax03/ElektReader/blob/f253e4b3457a58e4b7d5b25f6e8c255`
  `app/src/main/java/elektreader/view/TrimGUIImpl.java#L44`
- Uso di Stream e lambda per la generazione della view a lista.
  `https://github.com/pallax03/ElektReader/blob/b41a4af9c31d3d9ec4d1dec86bfdd7`
  `app/src/main/java/elektreader/controller/SongsController.java#`
  `L119`
- Creazione e uso di functional interface per generare shortcuts `https://`
  `github.com/pallax03/OOP23-ElektReader/blob/ac0f88d64b02635a612182f7b006fc5b`
  `app/src/main/java/elektreader/controller/GUIController.java#`
  `L361`

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

Alex: sebbene l'applicazione non sia completamente efficente, e purtroppo mancano alcune funzioni implementative, ritengo di aver suddiviso e implementato in maniera accettabile e funzionale con poche ripetizioni di codice, tutta la logica del programma. Ritengo di aver gestito in maniera ccettabile le diverse eccezioni, effettuando parecchi test, durante l'implementazione logica. Penso che la parte più complessa sia stato più l'aspetto progettuale e iniziale del progetto come l'importazione delle varie librerie e il primo approccio con le tecnologie utilizzate. Non mi ritengo però soddisfatto del progetto finale, voglio continuare a migliorarlo e aggiungerci funzionalità.

Christian Luca: sebbene sia abbastanza soddisfatto del mio operato, al tempo stesso sono consapevole del fatto che basare una catena di riproduzioni su indici interi possa non essere troppo efficiente, e che anzi possa da un momento all'altro causare problemi se la gestione di quest'ultimo non è stata pensata a 360 gradi. Ciononostante, il codice risulta essere d'altro canto abbastanza solido per via della massiccia presenza di opzionali atti ad evitare eccezioni di sorta che potrebbero verificarsi a run-time.
Per quanto riguarda il continuo del progetto oltre il raggiungimento del superamento dell'esame: potrebbe anche succedere poiché come scritto nell'incipit della nostra relazione, il progetto ci è stato in realtà realmente commissionato da un'associazione; pertanto, presumibilmente richiederà manutenzione supplementare col passare del tempo.

Samuele Casadei: mi ritengo abbastanza soddisfatto del mio lavoro, il codice è più o meno robusto del punto di vista dell'handling degli errori;

tuttavia, temo di aver lasciato troppa parte del lavoro all'utente per quanto riguarda l'ordinamento dei brani e mi rendo conto che sicuramente si poteva fare di meglio. Nonostante ciò, questo progetto è stato molto utile per mettere alla prova le mie capacità nella progettazione e comunicazione con i compagni. Credo proprio che ci sarà bisogno di ulteriore manutenzione anche dopo la consegna, ma non è assolutamente male per essere un primo approccio.

Matteo Zoli: nonostante sia evidente che non avessi la parte di progetto più sostanziosa e parte di essa fosse aggiungere funzioni in parti di codici dei miei compagni, mi ritengo abbastanza soddisfatto del mio lavoro, sebbene riconosca che non sia assolutamente perfetto. La cosa che mi da assolutamente più soddisfazione è stato riuscire a fare un lavoro, che ritengo abbastanza ben fatto, nella fase di ricerca delle librerie per eseguire l'operazione di taglio dei brani, poiché fin'ora riconosco di non aver mai fatto abbastanza ricerca prima di iniziare a scrivere codice.

Appendice A

Guida utente

Il programma inizialmente ha bisogno di caricare un ambiente (environment), da esso poi mostrerà graficamente se presenti delle varie playlist, ogni playlist conterrà almeno un brano riproducibile, selezionata una playlist, compariranno graficamente dei controlli media, essi saranno responsabili della riproduzione della playlist, sarà possibile poi visualizzare nel pannello a destra tutte le informazioni della playlist selezionata e tutti i brani contenti in essa, sarà possibile poi con un semplice click, la riproduzione di tale brano.
I controlli media sono responsabili della riproduzione della playlist, essa infatti è una coda di brani in ordine di indice, l'indice viene scelto in base al nome del file; infatti, vengono numerate (se esse non sono numerate verranno posizionate con il primo indice disponibile presente).
Viene resa disponibile la barra dei controlli media, appena viene selezionata una playlist poiché esse verranno riprodotte nel loro ordine definito naturale con gli indici calcolati, se l'utente poi vorrà una canzone in particolare potrà o andare avanti consumando la coda di riproduzione oppure selezionando direttamente con il mouse la canzone scelta. Il programma consumerà la coda fino al brano scelto, poiché finita una canzone continuerà a riprodurle fino a che la playlist non sarà terminata, se la playlist è terminata essa resetta la coda reimpostandola nel suo ordine naturale, oltre ai controlli base appena descritti, sono presenti anche funzionalità aggiuntive come:

- la possibilità di utilizzare una barra di velocità per impostare il brano in modalità lenta o veloce.
- La possibilità di controllare il volume della canzone che si sta riproducendo.
- La possibilità di stoppare la riproduzione quando finisce il brano che si sta riproducendo, senza passare alla canzone successiva in playlist.

- La possibilità di mettere in loop la canzone che si sta riproducendo essa verrà, riprodotta all'infinito fino a che il pulsante non sará disabilitato, appena verrà disabilitato la coda riprenderà normalmente nell'ordine naturale dalla playlist.

- La possibilità di riprodurre le canzoni in un ordine non naturale ma casuale.

- La possibilità di gestire la coda di riproduzione, con questo pulsante infatti verrà visualizzata la coda di riproduzione completa da quella che sta riproducendo fino all'ultima che verrà riprodotta, da qui infatti sarà possibile, modificare il loro indice, modificando il loro ordine di riproduzione trascinando con il mouse nella posizione dove si posiziona il cursore, essa non cambierà l'ordine naturale dei brani ma verranno solamente riprodotte in ordine differente, ma verrà poi resettato se si seleziona un'altra playlist.

- Sarà poi possibile trascinare le canzoni premendo ctrl sarà possibile selezionarne più di una, il programma poi è in grado di creare una nuova playlist richiederà il nome della playlist che si vuole modificare, se la canzone o le canzoni verranno trascinate nel pannello playlist, comporterà un ricaricamento di tutto l'environment per gestire eventuali, ambiguità come possibili playlist diventate vuote dopo lo spostamento di tali brani. Invece le canzoni possono essere anche selezionate e trascinate nel pannello per modificare il loro ordine naturale.

I pulsanti posti in alto al programma sono responsabili di:

- File: caricare un environment anche se è già stato caricato, verrà caricato un altro environment.

- View: modifica la visualizzazione delle canzoni, in due modalità: icone (mostrano solo il titolo la durata del brano, e la lista (mostra più informazioni come; indice, titolo, artista, durata, tipo di file.)

- Find: possibilità di cercare in automatico tra, playlist e canzoni, in base al pattern specificato:

  - Caratteri alfanumerici qualsiasi -> titoli o artisti
  - Numero intero -> canzoni totali (playlist), indice (canzoni)
  - hh:mm / hh:mm:ss / mm:ss -> durata totale (playlist), durata (canzoni)

- Trim: selezionato un brano dal file system (bottoni "Select Track" per scegliere dal file system e "Select track demo" per caricare un brano preimpostato) e inseriti i campi richiesti (inizio e termine del taglio e nuovo nome da dare al brano tagliato) lo riduce, ovvero il brano potrà essere ridotto di durata (es. Al posto di durare 1:20, inizierà a 0:20 e finirà a 1:00; quindi, la durata finale sarà di 40 secondi)
- Queue: mostra l'attuale queue di riproduzione dei brani. /item ?: non mostra niente verrà implementato in futuro si pensa per un tutorial.

Appendice B

Esercitazioni di laboratorio

B.1 alex.mazzoni3@studio.unibo.it

https://github.com/pallax03/OOP

- Laboratorio 07: https://virtuale.unibo.it/mod/forum/discuss.php?d=147598#p209327
- Laboratorio 09: https://virtuale.unibo.it/mod/forum/discuss.php?d=149231#p211285
- Laboratorio 10: https://virtuale.unibo.it/mod/forum/discuss.php?d=150252#p212600
- Laboratorio 11: https://virtuale.unibo.it/mod/forum/discuss.php?d=151542#p213927

B.2 giovanni.luca@studio.unibo.it

- Laboratorio 07: https://virtuale.unibo.it/mod/forum/discuss.php?d=147598#p209274
- Laboratorio 08: https://virtuale.unibo.it/mod/forum/discuss.php?d=148025
- Laboratorio 09: https://virtuale.unibo.it/mod/forum/discuss.php?d=149231