Corso di laurea in Ingegneria e Scienze informatiche – Università di Bologna

Corso di Programmazione ad Oggetti a.a. 2015-2016

Relazione del progetto “AudioPlayer”

Francesco Vanucci (0000652626)

1) Analisi del problema

L'applicazione realizzata ha lo scopo di gestire e riprodurre file in formato .wav.

L'applicazione permette di salvare tracce associandole a file in formato wav presenti sul computer e creare playlist aggiungendovi i brani salvati.

Una volta effettuato l'accesso all'applicazione tramite login l'utente ha modo di:

- Visualizzare i propri brani precedentemente salvati sul player
- Visualizzare le proprie playlist
- Salvare nuovi brani e, in caso già presenti, modificarne il file da riprodurre
- Creare playlist con i brani salvati, assegnandovi un nome specifico
- Riprodurre brani e playlist con la possibilità di stopparli, metterli in pausa e, in caso sia in riproduzione una playlist, spostarsi all'interno della stessa di brano in brano
- Eliminare brani e playlist dal sistema, con conseguente rimozione dei file

E' previsto un solo login alla volta, quindi vengono mostrati tutti i brani e tutte le playlist dell'utente sulla base di cosa desiderato.

Quando un brano viene rimosso o modificato le playlist contenenti il brano vengono aggiornate in modo da evitare incoerenze con il file wav indicato da uno specifico brano e le playlist che lo contengono.

Durante lo sviluppo del progetto si è cercato il più possibile di separare concettualmente model, view e controller. Le view sono tenute all'oscuro di qualsiasi dato gestito dal model e allo stesso modo il model ignora per cosa verranno effettivamente utilizzati i dati. I controller invece, specialmente L'APController, hanno una vista più "ampia" dei dati e gestiscono le comunicazioni tra vista e model accedendo al minor numero possibile di informazioni.

I componenti del JFrame principale sono classi ben distinte che estendono classi swing, in questo modo si è cercato di rendere la view il più modulare possibile.

2) Organizzazione dei packages

I package sono stati suddivisi tenendo conto del pattern MVC e di conseguenza dividendo i 3 packages principali “view”, “controller” e “model” in ulteriori packages per separare i differenti componenti.

I package sono quindi i seguenti

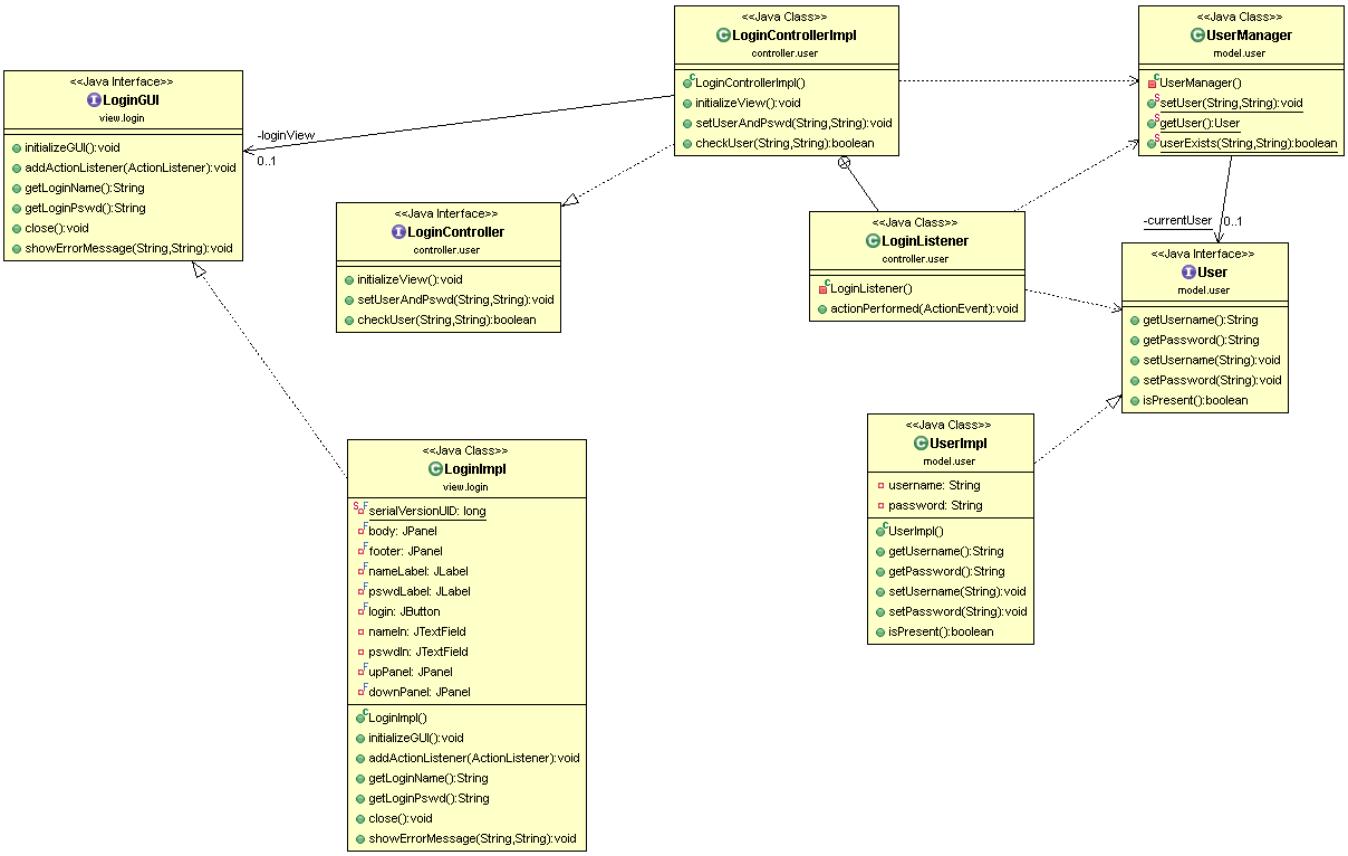
- Controller: E’ il package contente tutti i controller dell’applicazione, essi si occupano di gestire la parte “logica” dell’applicazione non direttamente collegata ai dati immagazzinati. E’ strutturato in modo da avere un controller per il JFrame principale e le opzioni selezionabili e un ulteriore controller per le tabelle dei dati, per l’aggiunta di brani e playlists e per il player audio.
  Il package è diviso in:
  - controller.user
  - controller.app
  - controller.adder
  - controller.data
  - controller.player

- Model: E’ il package contente tutti i dati dell’applicazione, così come le classi di accesso ai file
  Il package è diviso in:
  - model
  - model.track
  - model.playlist
  - model.user

- View: E’ il package contente tutte le finestre e i componenti visualizzabili, i componenti all’interno del frame principale sono stati separati a livello logico per una maggiore riusabilità e una migliore manutenzione
  Il package è diviso in:
  - view
  - view.create
  - view.data
  - view.login
  - view.player

3) Progettazione architetturale

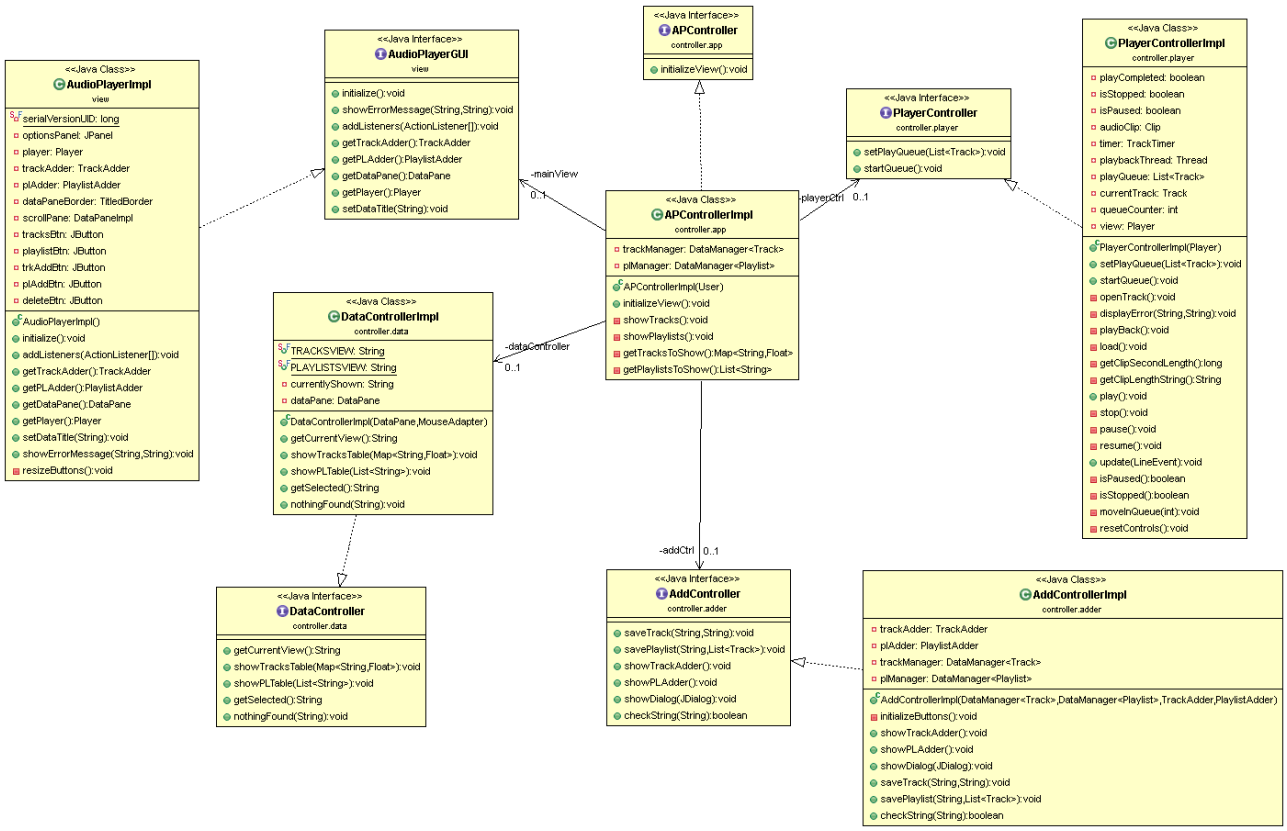
L'applicazione, tramite il main situato nella AudioPlayerApp, avvia il controller di login per l'accesso all'audioplayer



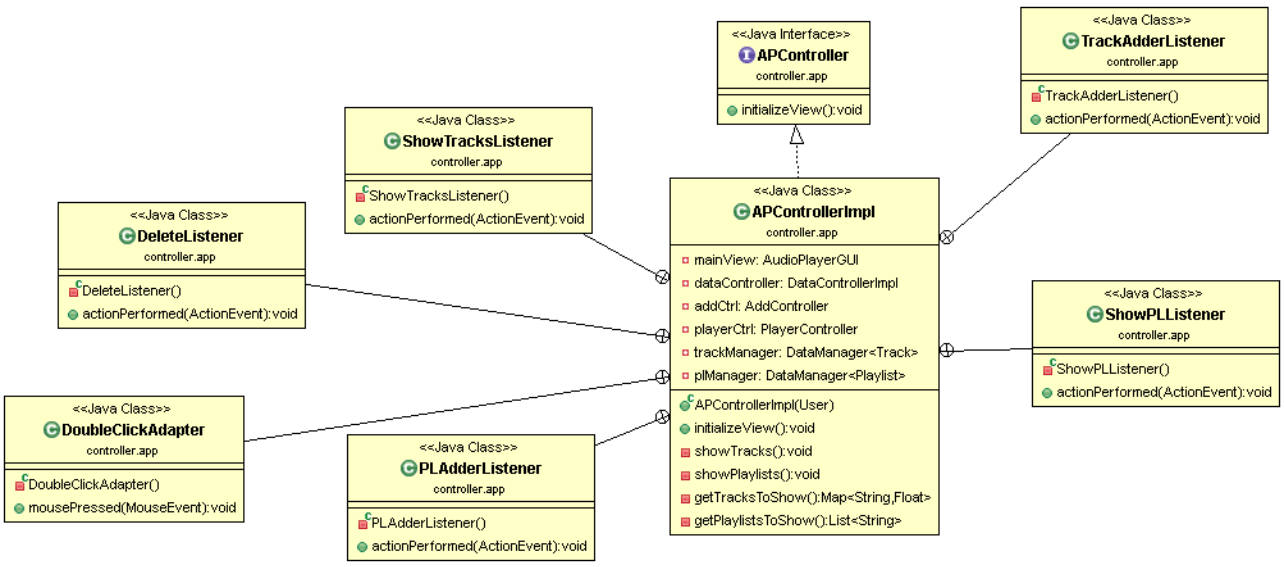
Il controller si occupa di visualizzare la LoginGUI, la quale permette l'inserimento dei dati utente che, alla pressione del tasto CONFERMA vengono gestiti dal controller che li setta tramite UserManager.

Per il Manager è stato usato il pattern Singleton per garantire che un solo utente sia istanziato al login.

Se il login ha successo viene chiamato il main controller per l'app e viene aperto il mediaplayer.

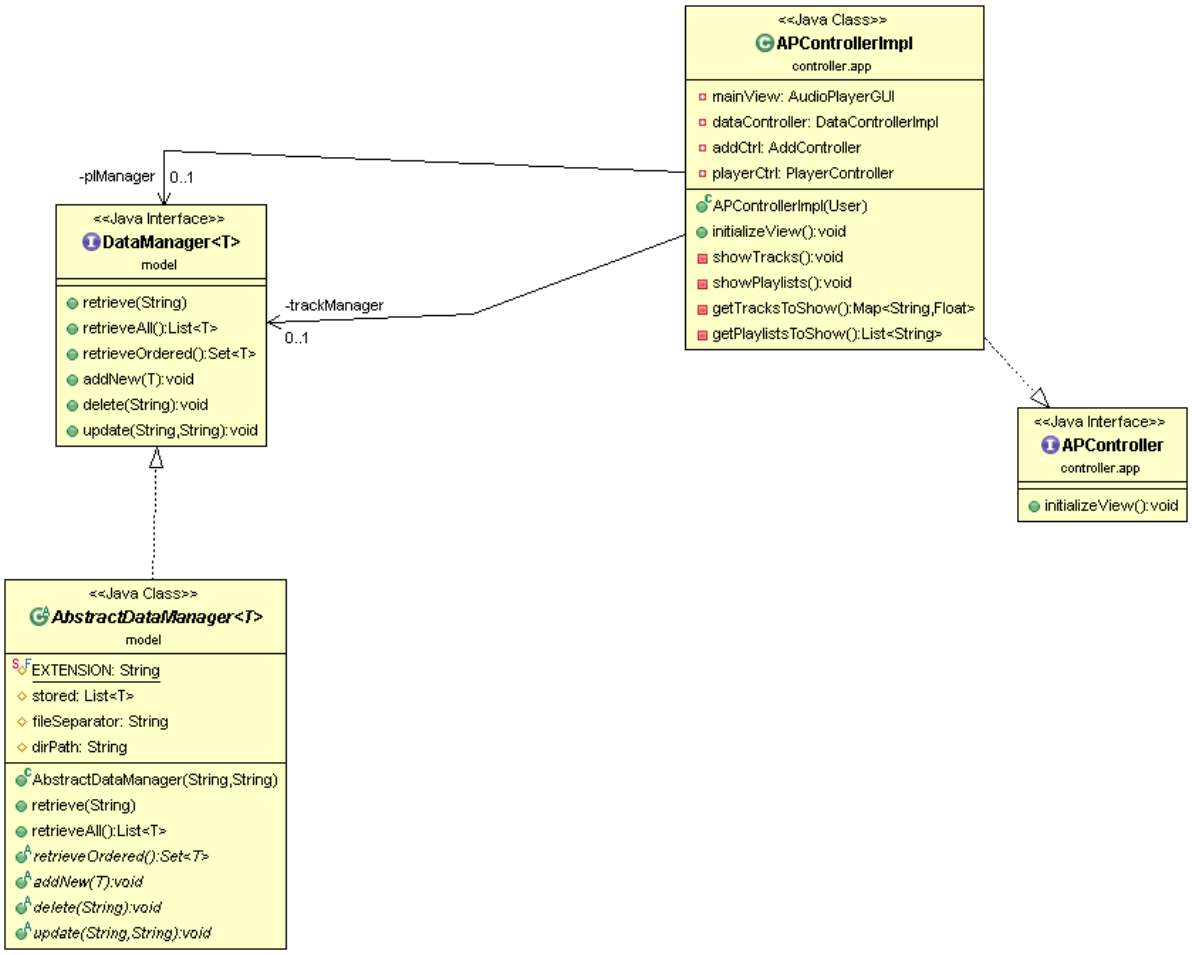


L'APController gestisce gli altri controller istanziandoli e definendo i listener dei vari controller.



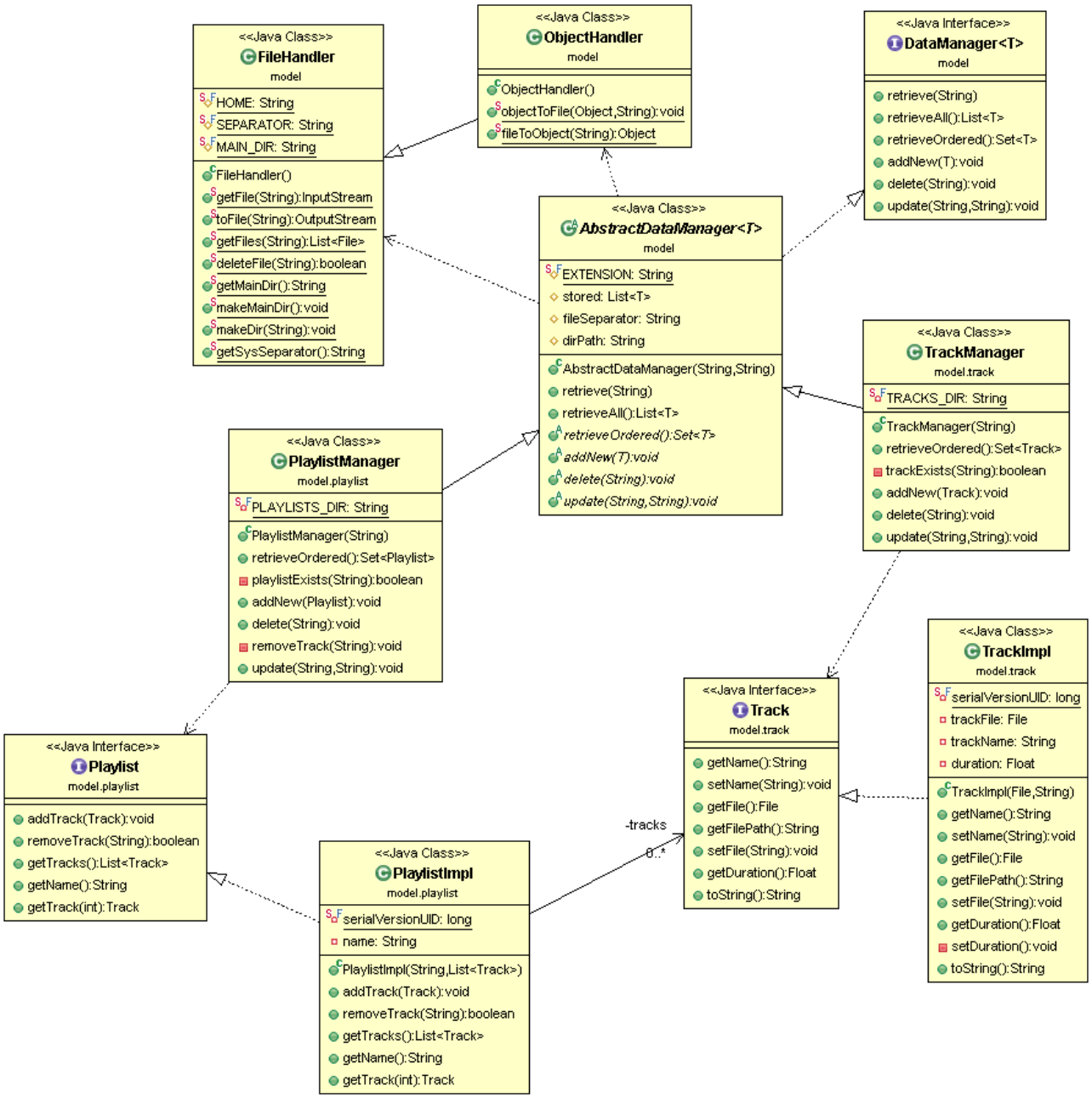
Con i listener il main controller si occupa di gestire le interazioni tra i controller e l'accesso ai DataManager per i dati, con l'unica eccezione del DataController a cui viene passato un riferimento ai manager per velocizzare l'accesso a tracce e playlist.

Nel dettaglio quindi il main controller instanzia i data manager per tracce e playlist per poi gestirne l'accesso.

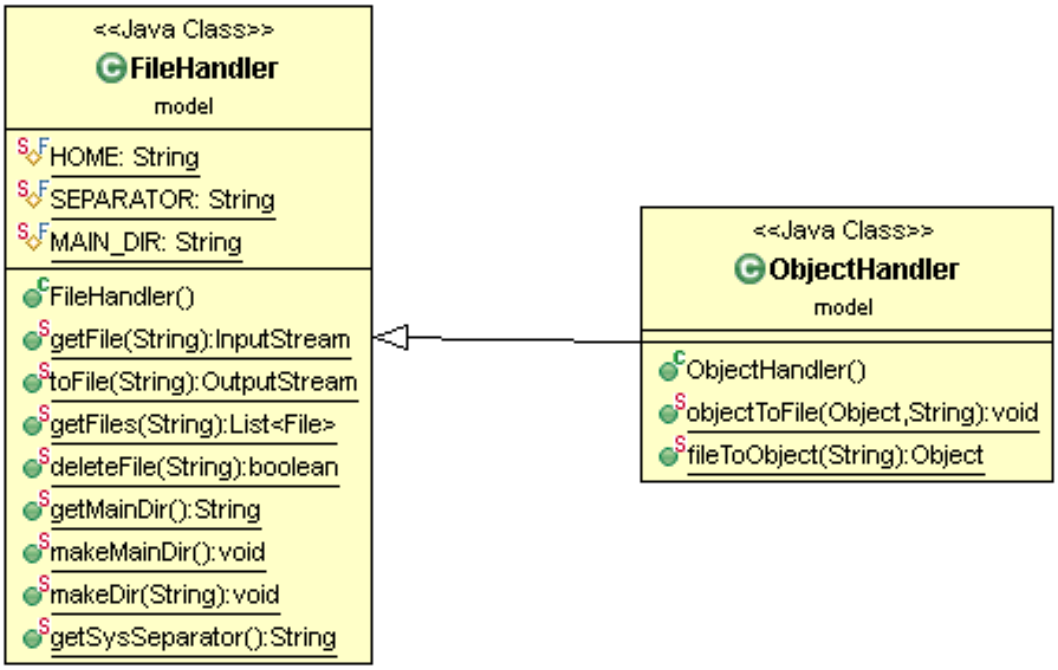


L'AbstractDataManager è una classe astratta estesa da due sottoclassi che si occupano dell'effettiva gestione di tracce e playlist. Si è utilizzato quindi il template method per collassare verso l'alto i metodi comuni ai due manager, differenziando poi le classi sulla base dei dati gestiti e metodi esclusivi.

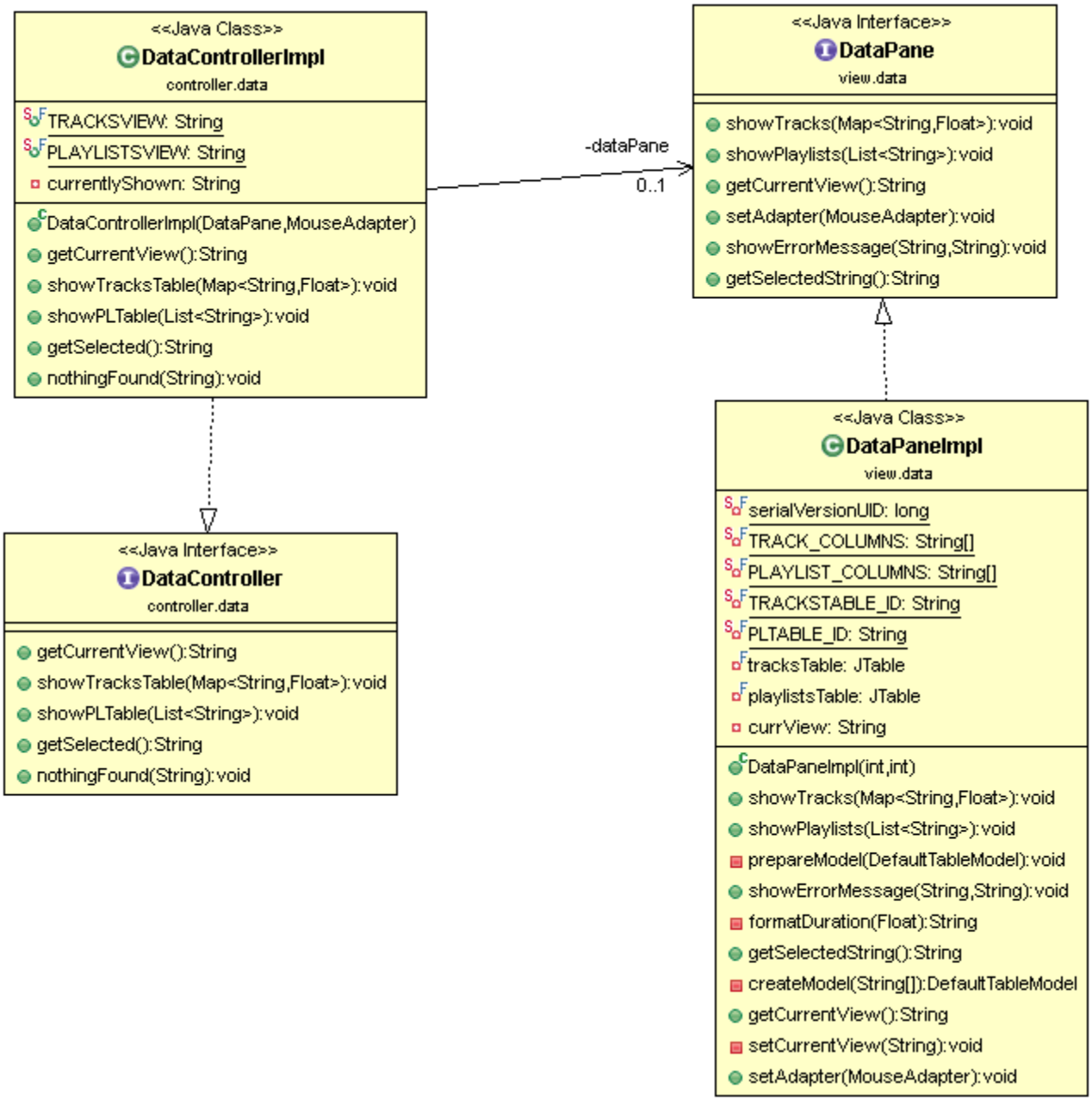
Di seguito il model del progetto, dove vengono gestite le tracce e playlist correnti, con metodi di aggiunta, recupero, recupero ordinato (con delegation) e rimozione. I manager operano sia sulle tracce e playlist correntemente nell'applicazione (poiché precedentemente recuperate o aggiunte) sia sui file delle stesse, aggiornando i dati ad ogni cambiamento.



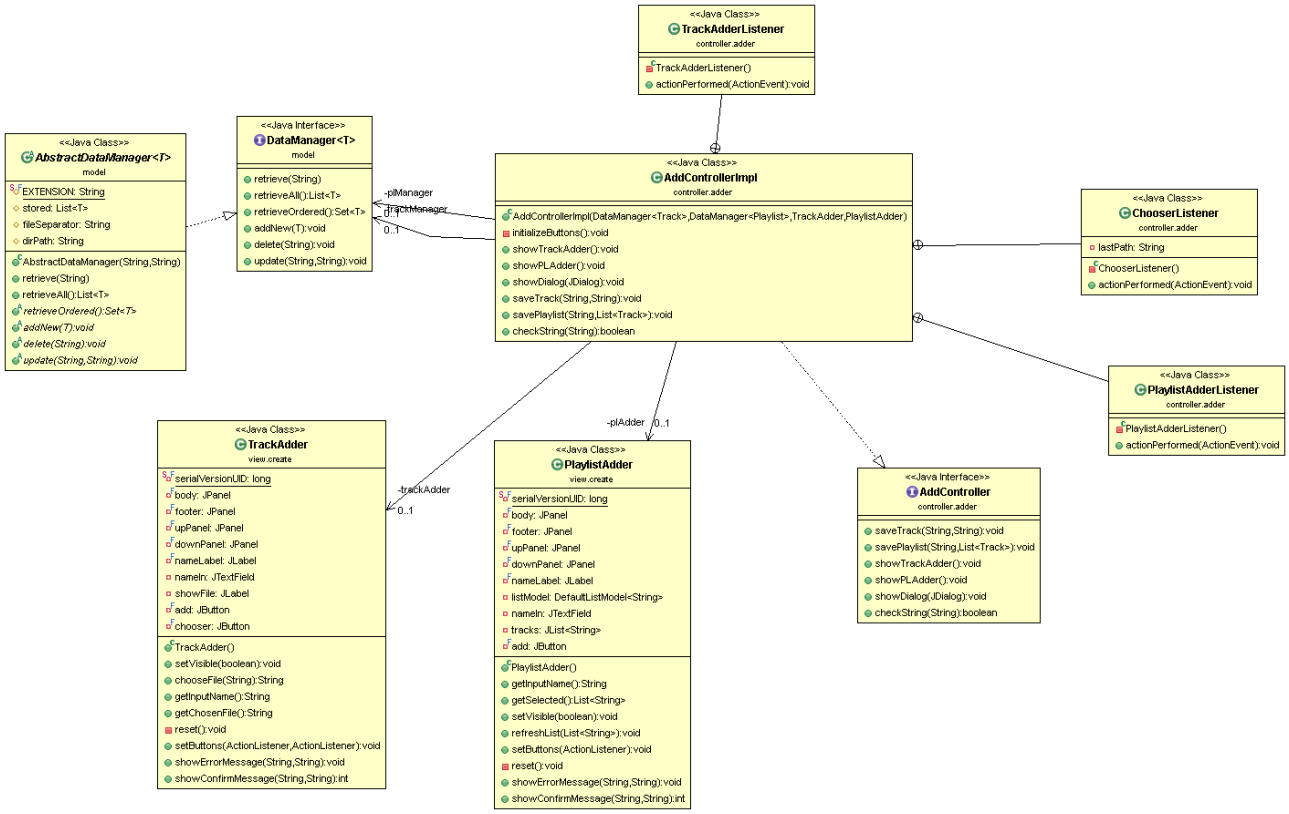
I manager utilizzano le classi statiche FileHandler e ObjectHandler per gli accessi su file standard e su file .dat relativi a oggetti. FileHandler e ObjectHandler sono classi di utility e pertanto si è deciso di definirle statiche. In questo modo è possibile anche impiegarle per mantenere informazioni come la directory principale dell'applicazione e garantendo una forma di accesso statica ai file nella directory dell'applicazione. Renderle classi istanziabili avrebbe portato solo ad un proliferarsi di oggetti che, a conti fatti, non sarebbero differiti in alcuna maniera in termini di accesso e delegando ai DataManager il compito di definire le sottodirectory specifiche all'interno della main directory.



Il DataController si occupa della gestione dei file da mostrare nelle tabelle dell'applicazione, e gestisce una view DataPane che estende JScrollPane e funge da contenitore delle tabelle di visualizzazione dei dati. Anche in questo caso il DataPane non ha accesso ai tipi Track e Playlist né tantomeno ai manager, Il DataController definisce la logica di visualizzazione con il DataPane che si occupa del mero popolamento delle table.



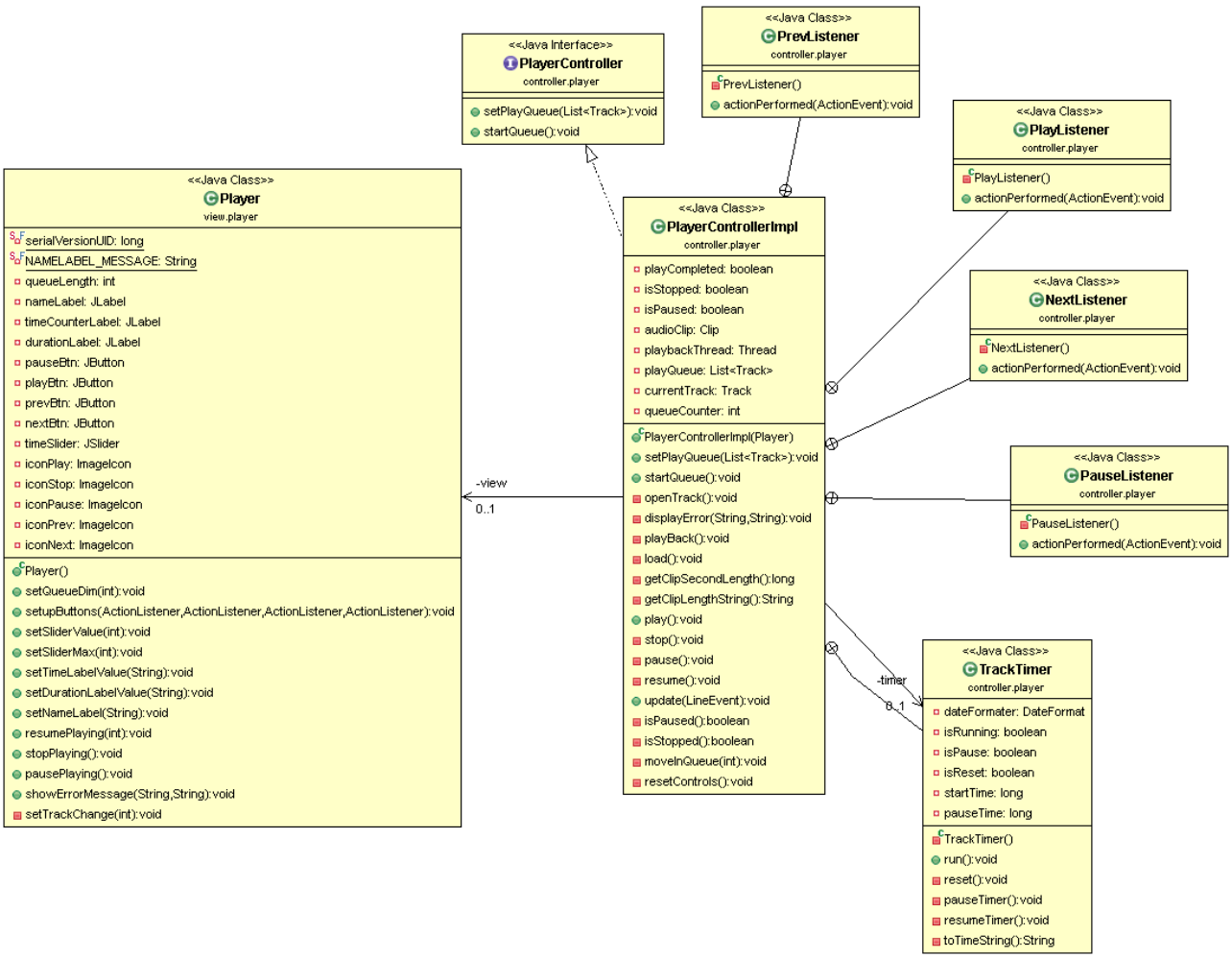
L'AddController si occupa invece di gestire i JDialog TrackAdder e PlaylistAdder e dell'aggiunta di tracce e playlist sulla base di quanto scelto dall'utente. In questo caso la classe AddControllerImpl ha accesso alle due istanze di tipo DataManager realizzate in TrackManager e PlaylistManager. ChooserListener e PlaylistAdderListener estendono ActionListener e si occupano di chiamare i metodi di salvataggio sul controller che delegano ai DataManager le operazioni effettive sui file. Questo è stato fatto mantenendo il concetto di separazione marcata tra vista delle informazioni tra Controller e Model



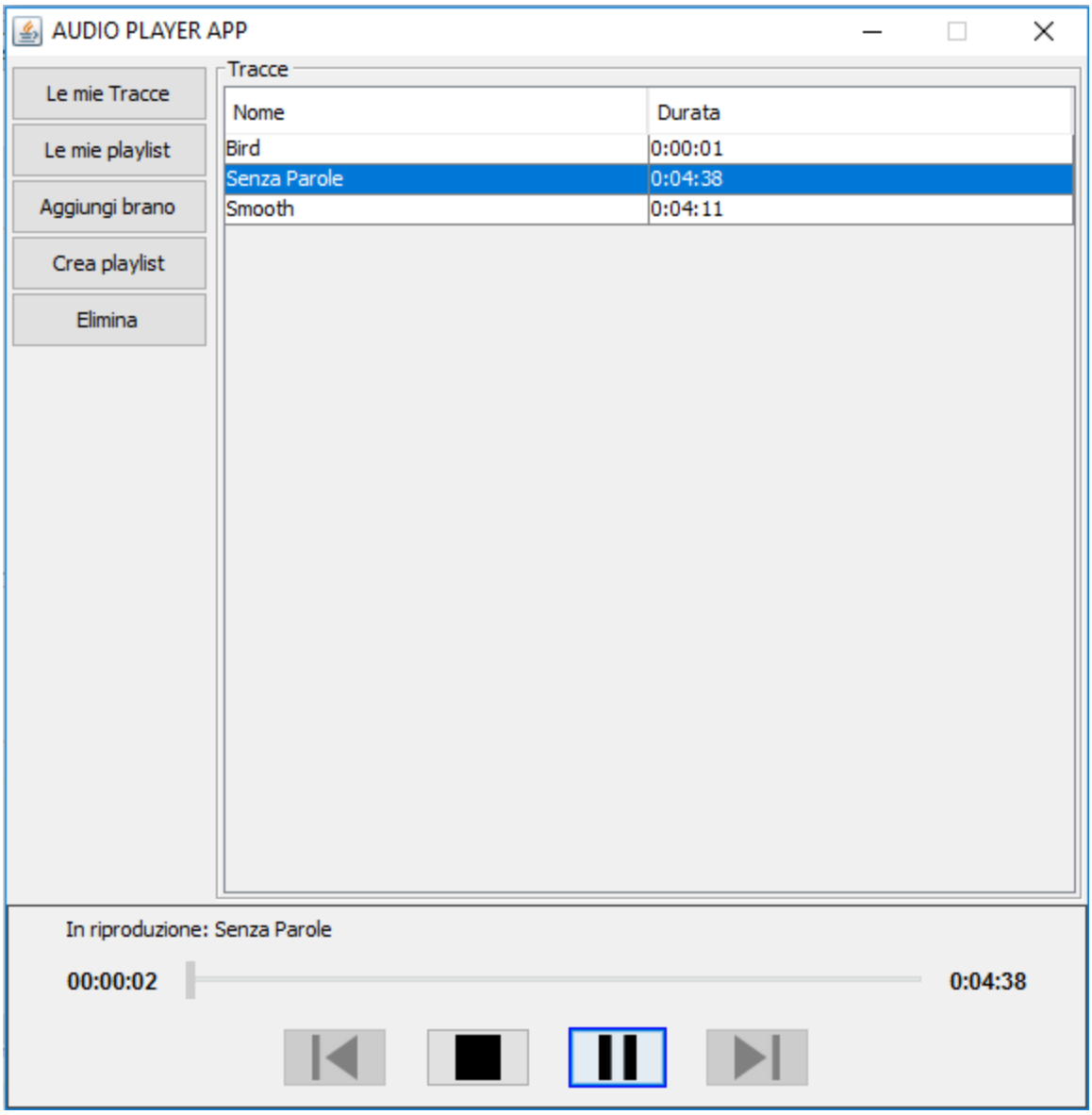
Il PlayerController si occupa della gestione della riproduzione dei singoli brani o delle playlist, è stato sviluppato a partire da un sample di player audio in Swing al seguente indirizzo:

http://www.codejava.net/coding/java-audio-player-sample-application-in-swing

L'applicato è stato integrato e modificato, rimuovendo dalla view ogni sorta di controllo e modificandone il layout, per il controller è stata gestita una queue di brani in modo da potersi muovere tra i brani a seguito della pressione dei tasti "previous" e "next", così come stoppare il brano, metterlo in pausa e riprenderne poi l'esecuzione. I flag di stato sono mantenuti nel PlayerController, mentre la view mantiene esclusivamente le informazioni da visualizzare e risponde alle richieste di modifica visiva. I Listener del PlayerController si occupano dei pulsanti del player a schermo, alla pressione del tasto Stop per esempio una chiamata al metodo stopPlaying() della view sostituisce la icon al momento visualizzata sul pulsante.



A livello di view l’applicazione si presenta in questo modo



4) Progettazione in dettaglio

- APController (interface) e APControllerImpl (concrete class): E' il main controller del sistema, è stato realizzato nell'ottica di avere un Controller che gestisse gli altri per limitare il più possibile il proliferarsi di accessi ai DataManager e di informazioni riguardanti l'utente in sessione. Questo controller comunica quindi col DataManager per recuperare informazioni su Tracce e Playlist per visualizzazione o riproduzione tramite chiamata al PlayerController. Contiene classi ActionListener per la gestione dei button di selezione e del double click. Nonostante l'alto numero di listener e in generale l'ampio scope della classe si è cercato di non renderla una superclasse ma delegare ai controller dei componenti compiti più specifici e di operazioni dirette sulle view di dominio. I Listener presenti nella classe APControllerImpl si rendono neccessari per l'accesso al Model e conseguente accesso ai dati.

- AddController (interface) e AddControllerImpl (concrete class): E' il controller che si occupa di salvare le informazioni riguardanti tracce e playlist richiedendo ai DataManager di effettuare le operazioni desiderate sui dati. Contiene classi ActionListener per la pressione dei bottoni di aggiunta. La classe AddControllerImpl ha come argomenti del costruttore le istanze TrackManager e PlaylistManager dichiarate da APControllerImpl, servendosi degli stessi per l'aggiunta di tracce e playlist.

- DataController (interface) e DataControllerImpl (concrete class): E' il controller che si occupa delle tabelle di visualizzazione dei dati, gestisce quale tabella mostrare e quindi passa le informazioni.

- PlayerController (interface) e PlayerControllerImpl (concrete class): E' il controller che si occupa della riproduzione dei brani in ogni suo aspetto, compresi i cambiamenti visivi da comunicare alla view. Una volta aggiunte le tracce nella queue queste vengono caricate solo all'occorrenza, questo in virtù del fatto che non tutte le tracce di una playlist potrebbero essere ascoltate e nell'ottica di playlist di grandi dimensioni questo potrebbe significare tempo di attesa inutile aspettando che tutte le tracce della playlist vengano inutilmente caricate. La classe contiene un metodo playback() che contiene un playbackThread() anonimo per il playback della

traccia. Presenta inoltre un TrackTimer che estende la classe Thread che si occupa di tenere il timing della riproduzione monitorando il playbackThread per eventuali interruzioni e lo stato isPause del controller per gestire un'eventuale pausa del timer. Aggiornandosi ogni 100 millisecondi, in caso di pausa la variabile pauseTime viene incrementata di 100 millisecondi in modo da conteggiare costantemente il tempo di pausa nel calcolo della durata di riproduzione raggiunta. Contiene inoltre classi ActionListener per la gestione dei pulsanti del player. Basa parte del suo codice sul player disponibile al seguente indirizzo: http://www.codejava.net/coding/java-audio-player-sample-application-in-swing

- LoginController (interface) e LoginControllerImpl (concrete class): E' il controller che si occupa del login, gestendo i dati inseriti e garantendo l'accesso all'audioplayer

- DataManager<T> (interface) e AbstractDataManager<T> (abstract class): E' un template per i manager di tracce e playlist, racchiude le operazioni comuni definendo contratti per i metodi da definire nelle classi concrete. Alcune operazioni hanno richiesto una gestione specifica da parte delle classi TrackManager e PlaylistManager. DataManager e AbstractDataManager fanno uso di generici concretizzati dalle classi TrackManager e PlaylistManager fungendo appunto da template generici per le stesse.

- TrackManager e PlaylistManager: realizzano l'AbstractDataManager e si occupano delle operazioni direttamente legate ai dati, gestendo caricamento, salvataggio e modifica sugli stessi. Il PlaylistManager in particolare presenta il metodo privato removeTrack() per la rimozione di una traccia in ogni playlist che la contenesse a partire dal nome della stessa.

- FileHandler (superclass) e ObjectHandler (specialization): Sono classi statiche il cui compito è garantire metodi di recupero e scrittura su file e object file così come definire l'indirizzo della main directory per l'applicazione sulla macchina corrente. Tutti i path dei file appartenenti all'applicazione vengono salvati con file relativi, utilizzando le costanti di

FileHandler per definire indirizzi “componibili” senza utilizzare mai indirizzi assoluti. Tutti gli indirizzi sono ottenuti a partire dall’utilizzo di metodi come System.properties().

- Track (interface) e TrackImpl (concrete class): Definisce l’oggetto traccia, sulla quale vengono memorizzati il nome e il file linkato (ovvero il file concreto da riprodurre) e la durata della traccia settata all’atto della creazione

- Playlist (interface) e PlaylistImpl (concrete class): Definisce una playlist contenente tracce, specifica un nome e una lista contenitore e viene aggiornata ogni qualvolta una traccia è stata editata e rimossa (tramite il PlaylistManager).

- UserManager: Definisce un manager per l’utente, controlla l’esistenza dello user nel file apposito e setta l’utente per la sessione corrente. E’ stato utilizzato il pattern singleton per garantire la presenza di un solo utente. E’ stato deciso di non memorizzare su file oggetti .dat per via delle poche informazioni contenute negli stessi, questo unito al fatto che solitamente non è previsto un sistema di login per un mediaplayer data la forte relazione con i contenuti locali dell’utente loggato nel Sistema Operativo e quindi è stato difficile fare riferimento ad un contesto reale.

- User (interface) e UserImpl (concrete class): Definisce una classe utente, utilizzata dal programma per filtrare le tracce e le playlist. Viene creata all’atto del login e utilizzata per costruire modularmente l’indirizzo delle directory di tracce e playlist.

- AudioPlayerGUI (interface) e AudioPlayerImpl (concrete class): E’ il JFrame principale dell’applicazione, contiene le altre viste dell’applicazione (esclusa la finestra di login iniziale). Utilizza BorderLayout con cui vengono distruibuiti i 3 componenti principali all’interno.

- LoginGUI (interface) e LoginImpl (concrete class): E’ la view di login e permette all’utente di inserire i propri dati.

- TrackAdder e PlaylistAdder: Sono JDialog e permettono di inserire le informazioni necessarie a creare tracce e playlist. TrackAdder utilizzare un JFileChooser per scegliere il file da inserire nella nuova traccia, PlaylistAdder mostra le tracce dell'utente permettendo una multiselezione delle stesse. Entrambi si compongono di 3 JPanel

- DataPane: Estende JScrollPane e si occupa di mostrare le tabelle contententi le info su tracce e playlist. L'effettivo posizionamento del DataPane viene gestito dal main Frame AudioPlayerImpl con BorderLayout.CENTER come orientamento, questo per garantire la massima espansione delle tabelle di visualizzazione.

- Player: E' la vista del player, basata sul modello presente al sito http://www.codejava.net/coding/java-audio-player-sample-application-in-swing come precedentemente notificato. Visualizza un JSlider per tracciare l'avanzamento del brano così come una rappresentazione fisica del minutaggio e il nome della traccia correntemente riprodotta; ovviamente sono presenti tutti i tasti di controllo del brano. Il codice presente sul link precedente è stato scelto in quanto sembrava un ottimo layout di partenza a cui applicare modifiche di design con tasti di controllo rivisti. Il Player carica le icone all'atto della costruzione e solo il pulsante di Play/Stop viene sostituito in itinere a seconda dello stato di riproduzione del brano, mentre i restanti pulsanti vengono abilitati e disabilitati sulla base delle indicazioni del controller. Per via dell'alto numero di elementi diversi è stato mantenuto il GridBagLayout.

ActionListener

L'applicazione fa largo uso di listener per gestire i vari pulsanti presenti, di seguito sono riportati i più rilevanti:

- TrackAdderListener e PlaylistAdderListener: Si trovano nella classe AddControllerImpl e ascoltano i pulsanti dei JDialog di aggiunta per aggiungere nuove tracce e playlist. In caso si aggiunga una traccia esistente è possibile sovrascrivere la precedente con conseguente chiamata al PlaylistManager per l'aggiornamento delle playlist. In caso si aggiunga invece una playlist già esistente l'aggiunta non viene permessa.

- DeleteListener: Si trova all'interno della classe APControllerImpl e si occupa di eliminare tracce o playlist selezionate. Il listener utilizza le informazioni del DataController per verificare la table su cui è stata eseguita la selezione, dopo di che chiama il manager opportuno per l'eliminazione.

- DoubleClickAdapter: Estende la classe MouseAdapter e si trova all'interno della classe APControllerImpl. Al doppio click su un brano o una playlist viene chiamato il PlayerController per preparare la queue. In caso si sia selezionata una playlist verranno recuperate e passate al PlayerController tutte le tracce all'interno della playlist.

5) Note e Autovalutazione

Come precedentemente sottolineato il programma non crea clip o copie di file in una cartella specifica, ma lavora con i file wav presenti nell'intero disco, questa decisione è stata presa per rendere il player più vicino ai mediaplayer tradizionali, tuttavia ha portato a diverse difficoltà all'atto della progettazione soprattutto in merito alla persistenza di modifiche.

Come già detto per il controller e la view del player è stato preso a riferimento il sito precedentemente indicato, che ho considerato un modello molto valido e che ho quindi integrato nel mio programma.

Nel metodo "retrieve" del DataManager<T> è presente un @SuppressWarning per via di un unchecked cast necessario per recuperare i dati.

Il progetto ha richiesto circa 110 ore ma non posso ritenermi del tutto soddisfatto. A fronte del rispetto di tutte le funzionalità non sono riuscito a realizzare un player per MP3 a causa del timore di non riuscire a incorporare nel tempo di lavoro uno studio di JavaFX, inoltre alcune scelte di design e modifiche in corsa hanno portato a perdere pulizia strutturale.