Alma Mater Studiorum Università di Bologna
Laurea in Ingegneria e scienze informatiche

Relazione
Groove & Pumpkin
Corso Programmazione ad Oggetti
A.A. 2014/2015

Matteo Gabellini
Alessandro Cevoli

23 Maggio 2015

Sommario

Questo documento è la relazione finale del progetto Groove & Pumkin svolto nell'ambito del corso di programmazione ad oggetti (laurea triennale in Ingegneria e scienze informatiche).
Lo scopo di questa relazione è illustrare le diverse fasi di sviluppo del progetto e di fornire una guida all'uso del programma finale.

Insieme alla relazione nel repository Bitbucket del progetto potete trovare i sorgenti del programma e il file eseguibile Jar.

Link al repository:
https://bitbucket.org/PumpkinHeads/progetto_oop

Indice

1 Analisi

1.1 Requisiti……………………………………………………………….4
1.2 Problema……………………………………………………………….4

2 Design

2.1 Architettura……………………………………………………………5
2.2 Design dettagliato…………………………………………………….6

3 Sviluppo

3.1 Testing Automatizzato ……………………………………………16
3.2 Divisione dei compiti e metodologia di lavoro………16
3.3 Note di sviluppo………………………………………………………17

4 Commenti finali

Conclusioni e lavori futuri……………………………18

Appendice Guida Utente……………………………19

Capitolo 1

Analisi

1.1 Requisiti

Il software mira a fornire la possibilità di riprodurre una playlist di brani musicali in formato midi e wave, sia in modalità classica che in modalità loop.
E' messa a disposizione anche una modalità shuffle che permette al termine di una traccia di scegliere in modo random la canzone successiva.
Il software permette anche di creare pattern ritmici attraverso una groovebox e di ascoltare anch'essi in modalità classica oppure in loop. Una volta che un groove è stato creato vi sarà anche la possibilità di salvarlo sul computer come file midi

La modalità loop rappresenta una modalità di riproduzione dove una volta che una traccia termina viene ripetuta per un numero indefinito di volte, fino a quando non è l'utente che esplicitamente cambia traccia nel caso del musicplayer o blocca la riproduzione.
Il Groove indica una serie ritmica composta da suoni di batteria riprodotti in un tempo predefinito all'interno del periodo dell'intera traccia. La Groovebox rappresenta lo strumento per definire queste serie ritmiche chiamate Groove.

1.2 Problema

Il music player dovrà essere in grado di gestire una playlist, cioè dovrà offrire la possibiltà di aggiungere brani e rimuoverli, dovrà offrire la possibilità di cambiare la traccia in riproduzione con la successiva, la precedente o con una qualsiasi della playlist.
Infine il music player dovrà offrire la possibilità di gestire lo stato della modalità loop e shuffle.
Nel music player le difficoltà saranno:
Gestire le due possibili modalità di estrazione delle varie canzoni dalla playlist.
Gestire la riproduzione di due formati musicali diversi.

La groovebox dovrà offrire tutti gli strumenti per definire un groove, riprodurlo e salvarlo come '.mid'.
La difficoltà nella groovebox sarà riuscire creare una sequenza midi partendo dalla rappresentazione di un pattern del groove definito con la groovebox.

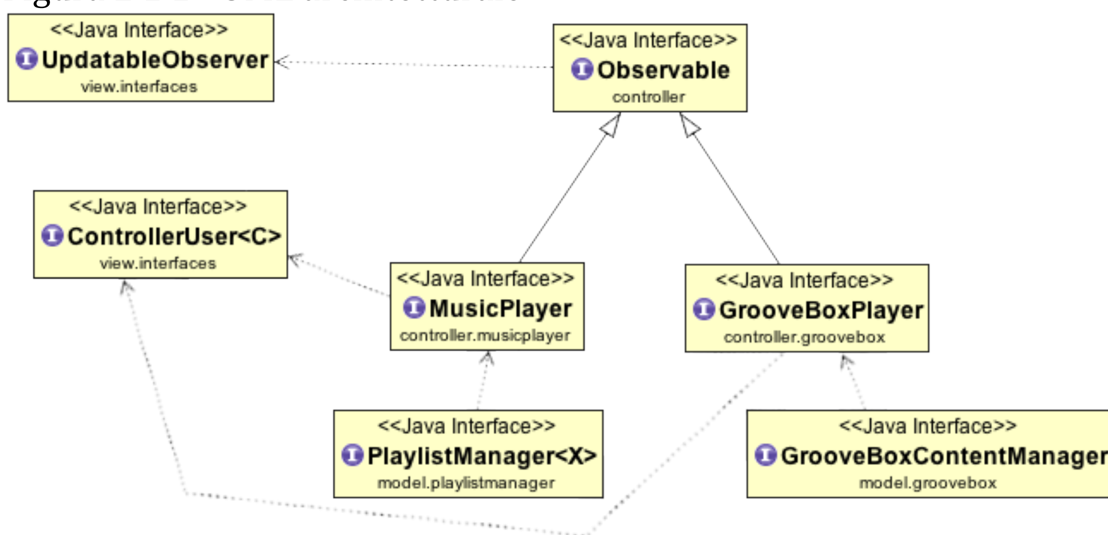
Il comparto grafico che gestisce la user interface dovrà essere in grado di ricevere i comandi impartiti dall'utente e inviarli in modo corretto al core dell'applicazione.

Capitolo 2

Design

2.1 Architettura

Figura 2.1.1 - UML architetturale



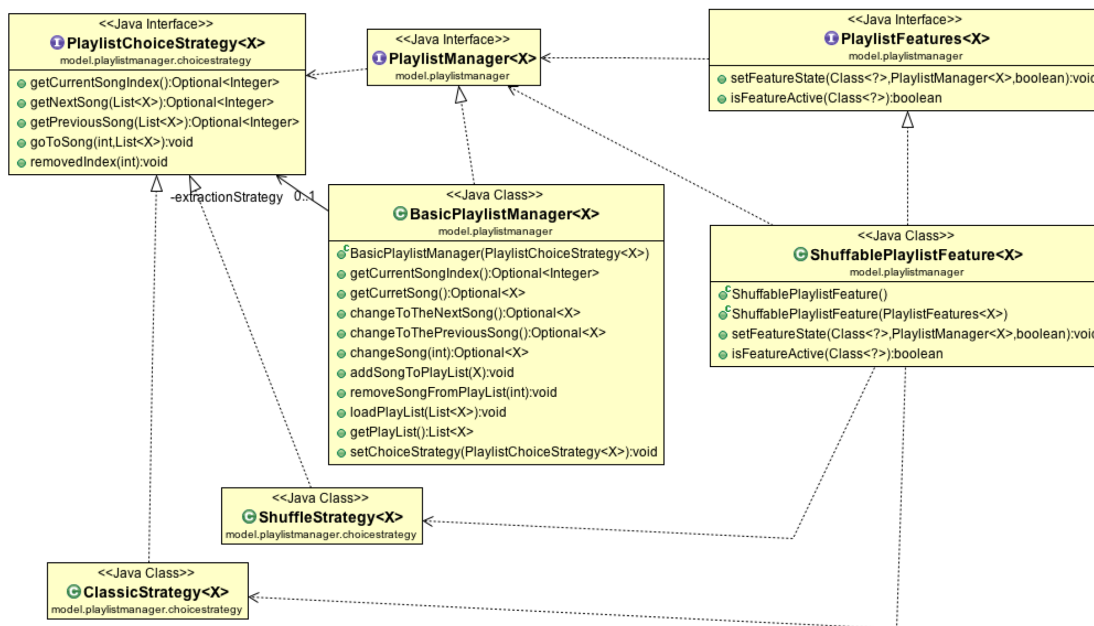
2.1.1 Architettura generale

In questo software un MusicPlayer e una GrooveBoxPlayer estendono un componente Observable, il quale possiede 0 – N riferimenti ad oggetti di tipo UpdatableObserver che rappresentano i possibili osservatori in ascolto degli eventi che accadono nei due player. Nel nostro caso gli osservatori sono i componenti della view, i quali si aggiornano in base alle notifiche inviate dal controller.
La user interface è esemplificata da ControllerUser che rappresenta degli oggetti che possono comunicare con uno dei due controller.
Un MusicPlayer per gestire una playlist ricorre ad un PlaylistManager mentre un Grooveboxplayer per gestire i groove creati utilizza un GrooveBoxContentManager.

2.2 Design dettagliato

2.2.1 Model

Figura 2.2.1.MusicPlayer - UML model music player:



Per quanto riguarda il model del music player come si può vedere è composto alla base da un generico PlaylistManager il quale si occupa della gestione base della lista dei brani, mentre per gestire l'estrazione dei brani da riprodurre ricorre ad un PlaylistChoiceStrategy.

2.2.1.a Pattern strategy:

Qui l'uso del pattern strategy è il risultato del ragionamento effettuato sul fatto che un PlaylistManager potrebbe eseguire le operazioni di estrazione dei brani dalla playlist seguendo logiche differenti.
Si può notare dallo schema UML(figura 2.2.1.MusicPlayer) che sono state implementate due logiche:
una che riguarda la modalità classica mentre una riguarda la modalità shuffle.

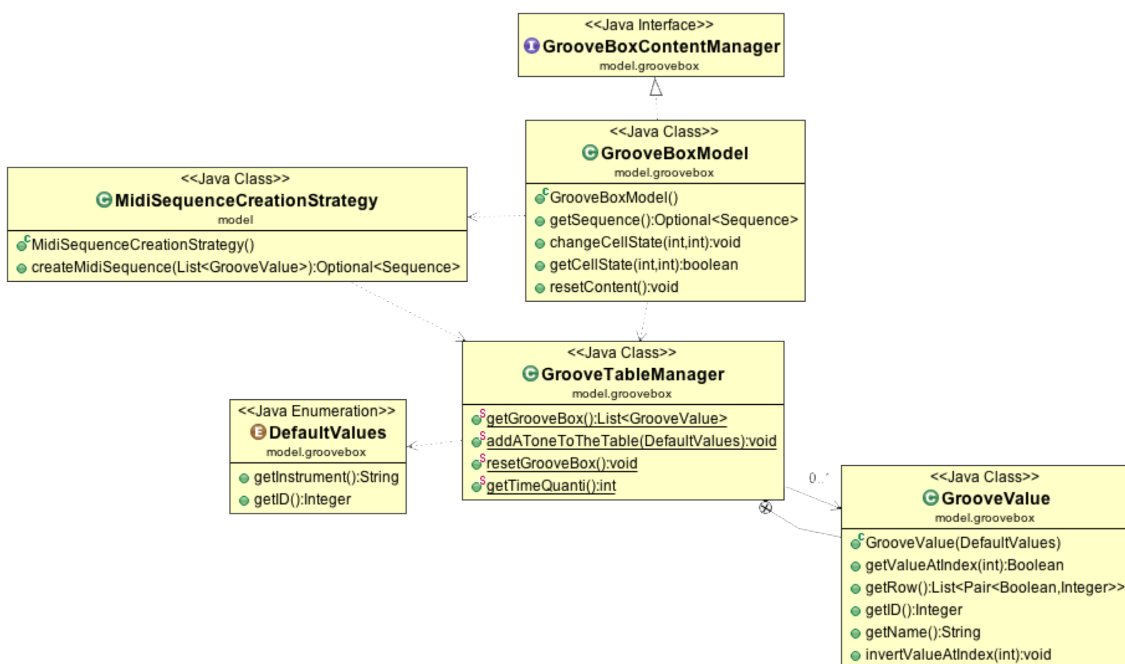
2.2.1.b Pattern Chain of Responsability:

Ragionando sul concetto di shuffle si intuisce che esso rappresenta una delle funzionalità aggiuntive che un PlaylistManager potrebbe possedere.
Considerando anche il fatto che un PlaylistManager potrebbe avere contemporaneamente più funzionalità aggiuntive si è deciso di modellare tali aggiunte secondo il pattern Chain of Responsability, perché garantisce una maggior

indipendenza tra il gestore della playlist e le classi che gestiscono le funzionalità aggiuntive.
A tal proposito è stata definita un'interfaccia PlaylistFeature che rappresenta una generica aggiunta di funzionalità. Le classi che la implementano, al loro interno, hanno un riferimento opzionale ad un'altra PlaylistFeature, tale riferimento permette di creare una sorta di catena di feature e quando un oggetto riceve una richiesta di gestione di una feature, se in grado di soddisfarla, se ne occupa lui. Altrimenti reindirizza la richiesta alla feature successiva di cui ne possiede il riferimento.

All'interno del software è la feature ShuffableFeatureManager che a seconda che lo shuffle sia attivo o meno, assegna al PlaylistManager la strategia di estrazione corretta da usare.

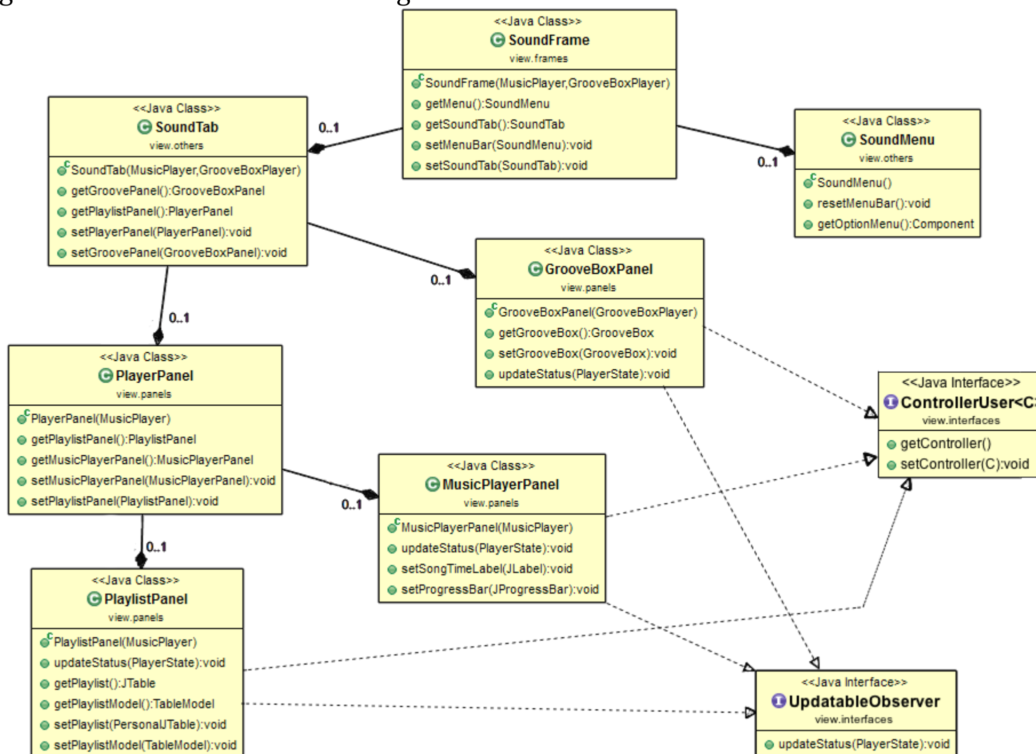
Figura 2.2.1.GrooveBox - Uml model GrooveBox:



Per quanto riguarda la Groovebox, alla base troviamo un GrooveBoxContentManager implementato da GrooveBoxModel. Per eseguire le operazioni esso ricorre a GrooveTableManager che racchiude al suo interno la rappresentazione, modellata attraverso elementi di tipo GrooveValue, della tabella utilizzata per definire i Groove.
GrooveBoxModel, per creare una sequenza riproducibile da un Sequencer Midi, ricorre a MidiSequenceCreationStrategy.
DefaultValues, infine, definisce la relazione tra le note e il loro corretto valore dello standard Midi ciò garantisce il corretto mapping tra la struttura interna e i dati riproducibili dal Sequencer Midi.

2.2.2 View

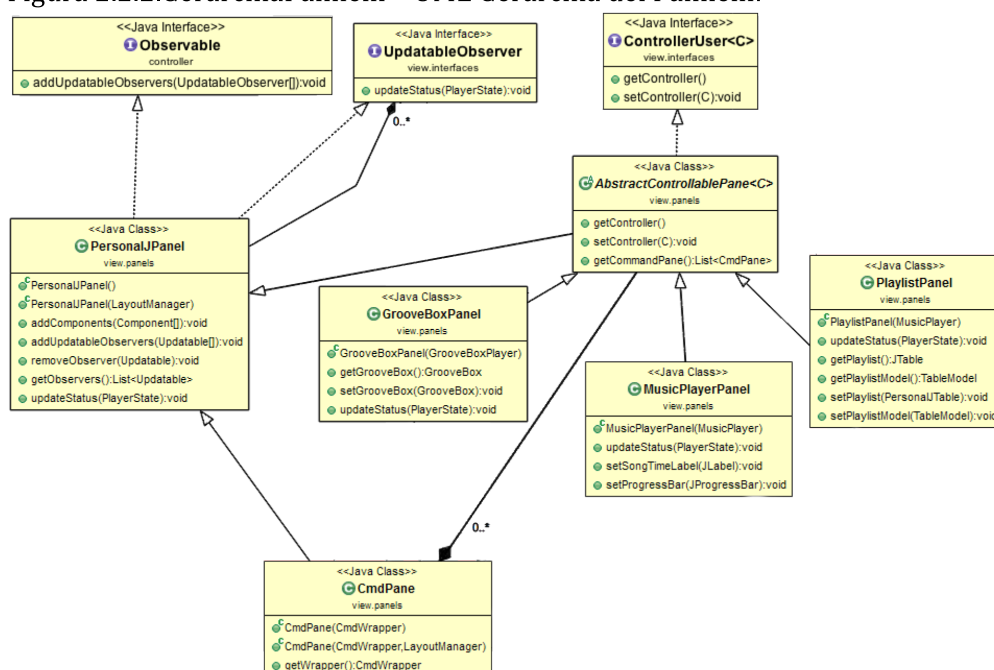
Figura 2.2.2.ViewGenerale – UML generale view:



La View si compone di un frame principale in cui vengono allocate un menù e delle tab rappresentanti i pannelli di comando per il music player e la groovebox player.

2.2.2.a Gerarchia dei pannelli:

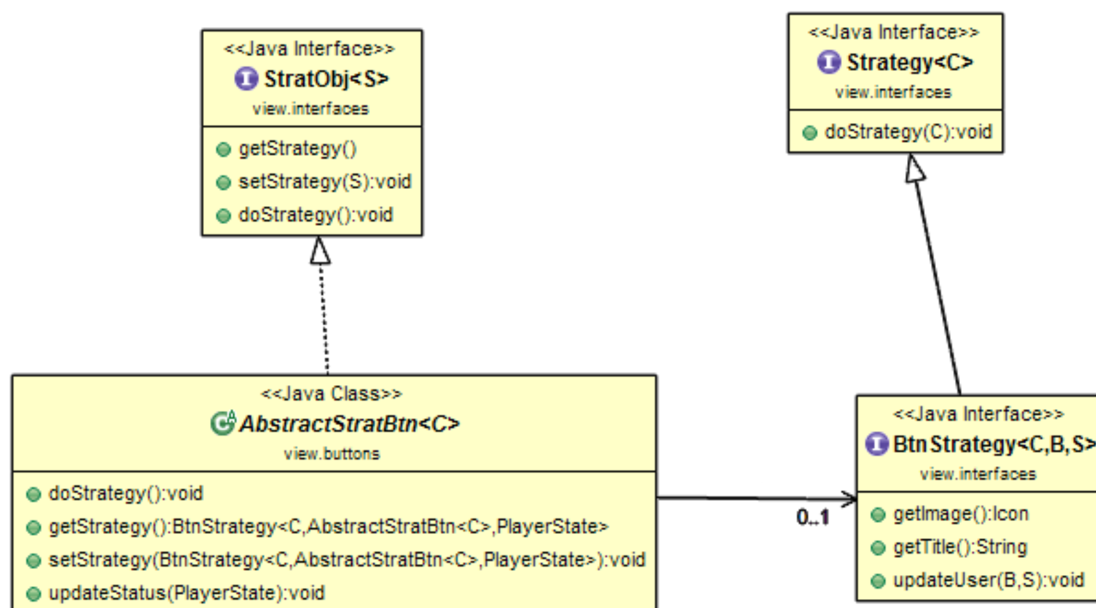
Figura 2.2.2.GerarchiaPannelli – UML Gerarchia dei Pannelli:



Come si può osservare dall'UML(Figura 2.2.2.ViewGenerale) dell'architettura generica della View, gli unici elementi che realmente sono in grado di 'intercettare' i messaggi e di interagire con il Controller sono i pannelli.
Descriviamo quindi come si è deciso di strutturare le classi che implementano la parte principale della GUI.
Senza entrare troppo nei dettagli, la ragione per cui è stata fatta tale scelta è per ridurre al minimo il numero di chiamate da effettuare tra i vari elementi in ascolto.
Come infatti si può notare da questo UML(Figura 2.2.2.GerarchiaPannelli) i pannelli stessi al loro interno adottano un pattern Observer per comunicare con le sotto componenti (i bottoni in primis, che verranno trattati in seguito); implementare questo pattern in ogni classe della View sarebbe stato inutile ed uno spreco di risorse.
Per la costruzione dei pannelli principali che gestiscono l'interfaccia del Music Player, della Playlist e della Groovebox, si è deciso di implementare un pattern Template Method. Le classi che sfruttano tale pattern sono l'AbstractControllablePane, che raccoglie le funzioni comuni delle 3 classi e implementante l'interfaccia ControllerUser, e i 3 pannelli sopra menzionati. I pannelli ad utilizzo generico utilizzano invece un PersonalJPanel. In tale modo si è cercato di scindere nel miglior modo le classi ad uso generico (per esempio quelle usate per il riempimento di un pannello) da quelle in cui interessava avere una implementazione particolare, con funzioni peculiari solo ad essa.

2.2.2.b Button Strategy

Figura 2.2.2.StrategiaBottoni – UML del design della strategia dei bottoni

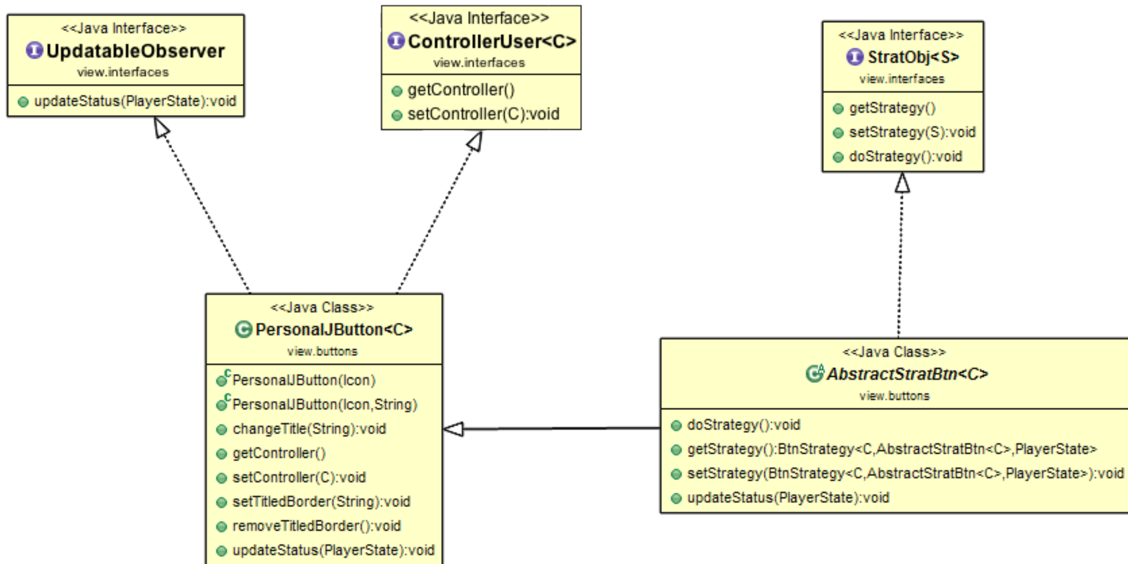


Trattiamo ora come si è deciso di gestire le varie funzionalità rese disponibili dal controller per renderle gestibili a livello utente.
A tal scopo è sembrato opportuno sfruttare un pattern Strategy applicandolo ad una famiglia di bottoni funzionali, in modo che le funzionalità fossero totalmente indipendenti dal bottone e quindi solo specifiche della strategia\e. Anzi sia la strategia stessa a definire il bottone!

Osservando l'UML(Figura 2.2.2.StrategiaBottoni) si notano le interfacce Strategy, che identifica il tipo di oggetto che stiamo usando (ovvero una strategy), e ButtonStrategy che invece identifica un tipo di strategia applicabile ad un bottone.
I bottoni utilizzati implementano l'interfaccia StrategicalObject che rappresenta un oggetto a cui è possibile associare una strategia e fornisce le azioni principali applicabili su quell'oggetto che "wrappa" la Strategia data.

Si allega inoltre un piccolo UML che esemplifica la struttura gerarchica dei bottoni.

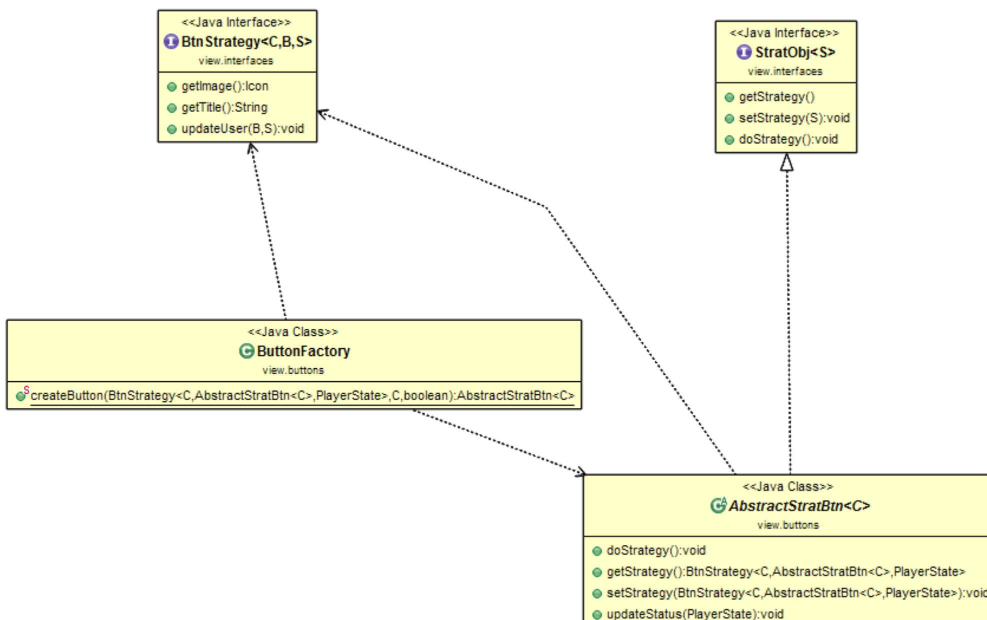
Figura 2.2.2.GerarchiaBottoni:



Ogni bottone funzionale è definito tramite una classe concreta PersonalJButton che ne definisce i dettagli di margine e caratteristiche comuni a livello grafico. Ogni bottone è un possibile UpdatableObserver per un oggetto di tipo Observable, ed inoltre ha la possibilità di "wrappare" un controller (su cui può venire definita la strategia).

2.2.2.d ButtonFactory

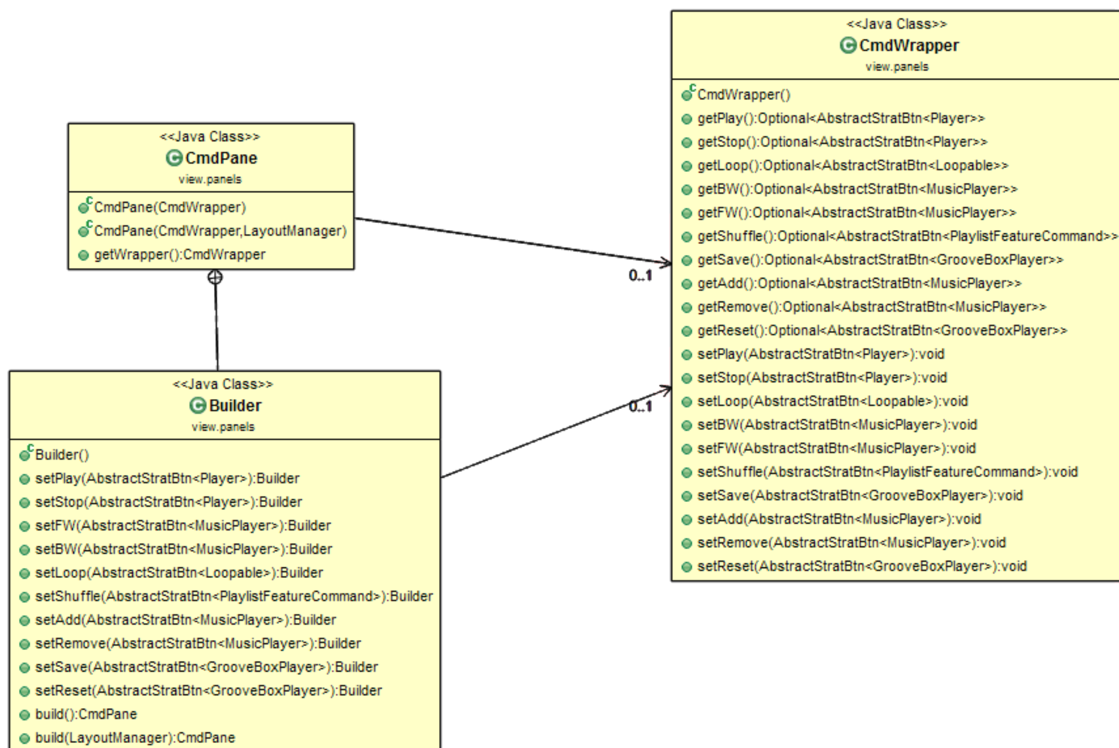
Figura 2.2.2.FactoryBottoni – UML del factory dei bottoni



Per la creazione dei bottoni funzionali ci si è affidati all'utilizzo di una Simple Static Factory poiché si è deciso di non fare uso di un'implementazione concreta dei bottoni, bensì crearne di anonimi attraverso la classe astratta.
Inoltre in questo modo la gestione delle peculiarità di alcuni bottoni vengono direttamente gestite all'interno della factory, senza andare ad implementare delle intere classi, o crearne una che le gestisca tutte.

2.2.2.e CommandPane e Pattern Builder

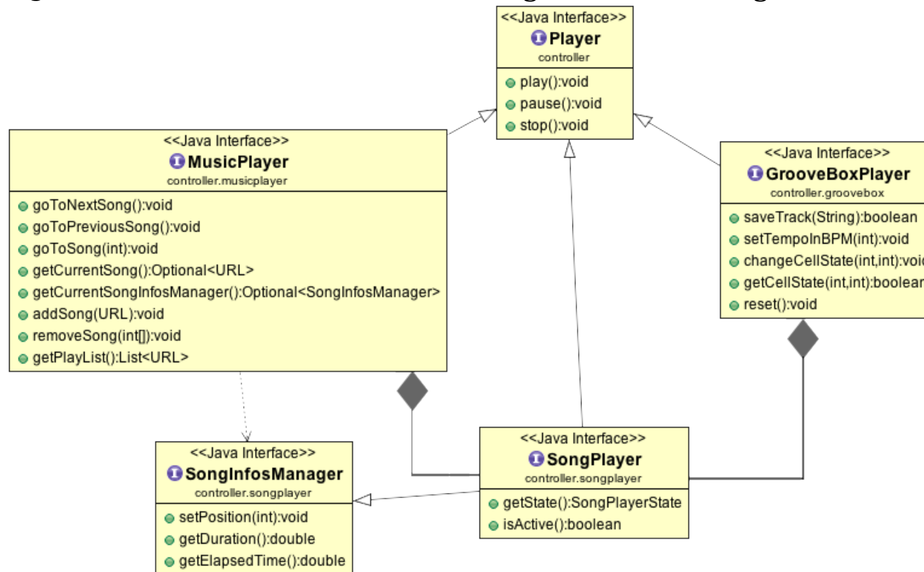
Figura 2.2.2.PatternBuilder:



Con questa classe si è voluto fornire un metodo semplice per costruire un pannello che dovesse contenere una fila di bottoni tra quelli disponibili all'interno del software, per poi essere aggiunto all'interno di un pannello più grande.
Per fare ciò oltre al pattern Builder, usato per popolare il CommandPane, ci si è affidati all'uso di una classe che funzionasse da container dei bottoni, così che, oltre ad essere "wrappati" all'interno del Component, fossero anche reperibili direttamente tramite relativi getter.

2.2.3 Controller

Figura 2.2.3.ControllerGenerale – UML generale del design del controller

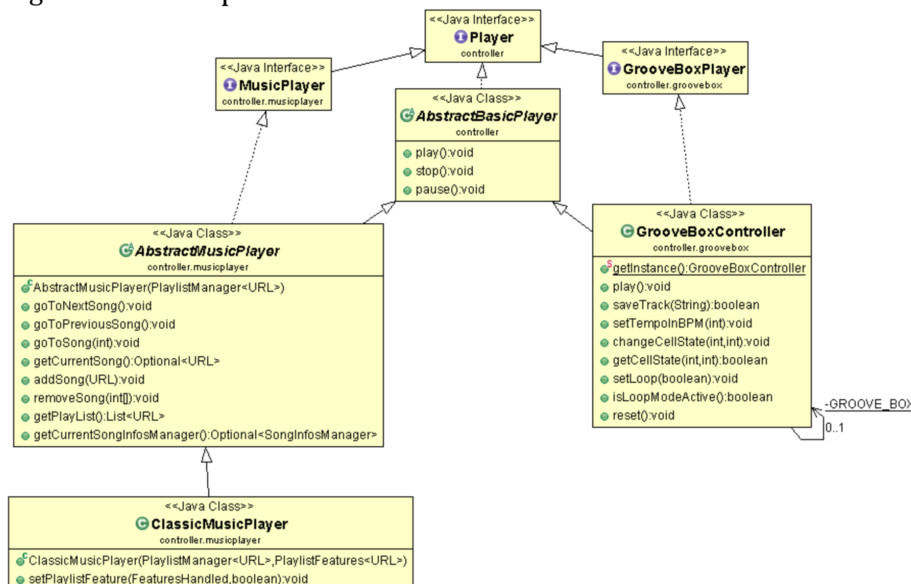


Il controller si compone di un Player che si occupa della gestione dei comandi.
Le funzioni del Player a loro volta sono arricchite da un MusicPlayer che aggiunge i comandi per la gestione e navigazione di una playlist. Altre funzionalità di un player vengono aggiunte da un GrooveBoxPlayer il quale rende disponibili i comandi di gestione di una groove box.
Sia il MusicPlayer che il GrooveBoxPlayer si compongono di un elemento di tipo SongPlayer il quale riproduce concertamente la musica

2.2.3.a Uso Pattern Template Method

Per la realizzazione del controller è stato utilizzato il pattern template method:

Figura 2.2.3.TemplateMethod:

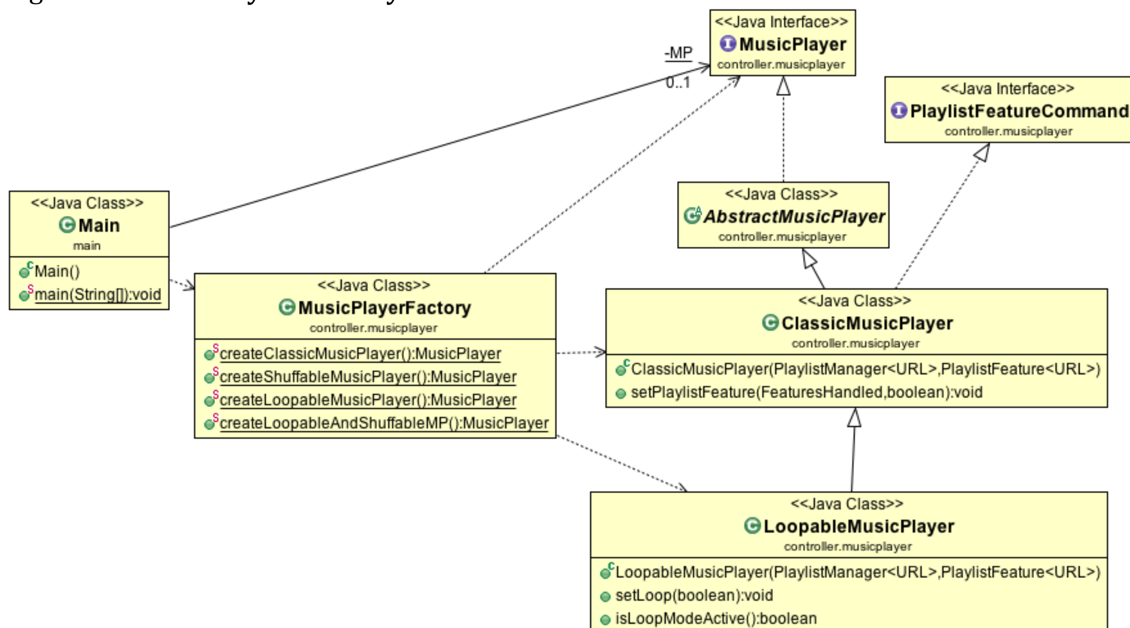


E' stata sviluppata una classe astratta che implementasse le operazioni base di un player.

Successivamente tale classe viene concretizzata con GrooveBoxController per quanto riguarda la groove box, mentre per quanto riguarda il music player la classe è stata ulteriormente estesa da un'altra classe astratta che implementa le operazioni base di un MusicPlayer per poi essere concretizzata con ClassicMusicPlayer.
Le ragioni che hanno portato all'uso di questo pattern nascono dal fatto che le funzionalità definite player potevano essere implementate allo stesso modo sia per il music player che per la groovebox l'unica cosa che le differenzia è il modo in cui caricano i brani da riprodurre perciò tale aspetto viene implementato concretamente in AbstractMusicPlayer e in GrooveBoxController

2.2.3.b Uso Pattern static Factory

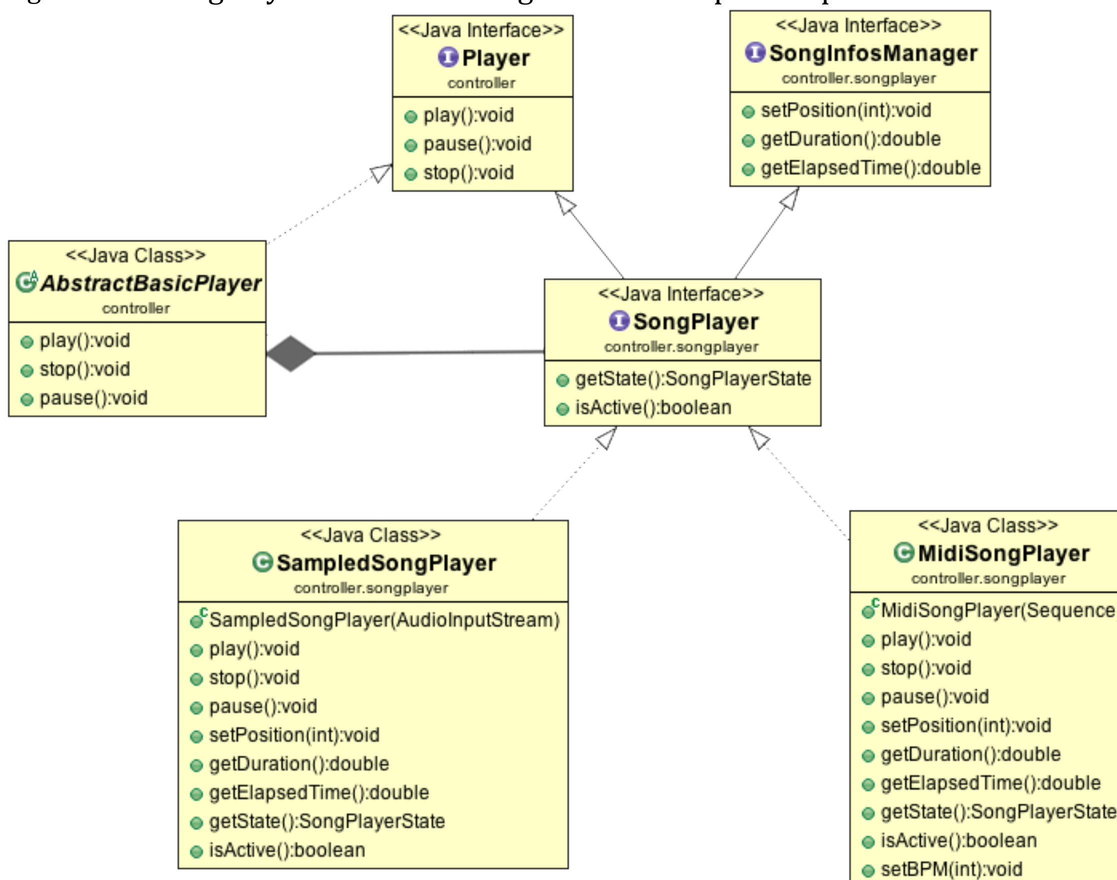
Figura 2.2.3.FactoryMusicPlayer



Dato che possono esistere diversi tipi di Music Player con funzionalità differenti si rischiava di avere una proliferazione di metodi in diverse classi che potevano anche risultare difficili da usare. Per questo motivo è stato modellato un MusicPlayerFactory il quale espone dei metodi statici per la creazione di diversi tipi di MusicPlayer.
Saranno tali metodi, al loro interno, a gestire la corretta creazione del MusicPlayer in base alle funzionalità desiderate.

2.2.3.c Design classi che riproducono musica

Figura 2.2.3.SongPlayer – UML del design delle classi per la riproduzione musicale:



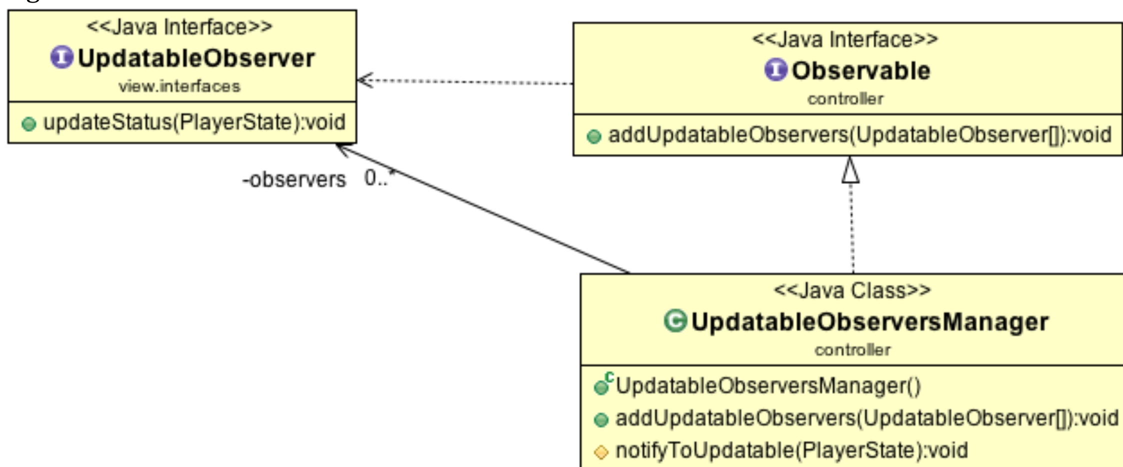
Per quanto riguarda la riproduzione concreta della musica è stato deciso di strutturarla come mostrato nell'UML(Figura 2.2.3.SongPlayer). L'interfaccia SongPlayer rappresenta un generico riproduttore di un brano. Tale classe è implementata da SampledSongPlayer e MidiSongPlayer.

La prima racchiude al suo interno la logica per la riproduzione della musica formato "wave" mentre la seconda racchiude la gestione della riproduzione Midi.

Tale modellazione permette al AbstractBasicPlayer di inviare i comandi ad un SongPlayer senza preoccuparsi di che tipo è realmente.

2.2.4 Pattern Observer tra View e Controller

Figura 2.2.4.PatternObserver



Per gestire la comunicazione del cambio di stato dei player, da parte del controller verso la view, è stato deciso di modellare delle classi secondo il pattern observer.
Un'interfaccia Observable definisce un metodo per aggiungere uno o più osservatori ad un oggetto che implementa tale interfaccia.
Observable è implementata nel nostro software da UpdatableObserversManager che racchiude al suo interno la gestione degli osservatori e fornisce un metodo protected per eseguire la notifica ad essi.
UpdatableObserversManager viene estesa dalle classi del controller del music player e della groovebox.

Capitolo 3

Sviluppo

3.1 Testing Automatizzato

Sono stati sviluppati test automatizzati che sfruttano la suite JUnit, per testare le varie parti del music player.
Il primo controlla il corretto funzionamento delle operazioni sulla playlist effettuando inserimenti, cancellazioni e navigazione tra i brani.
Il secondo test mette alla prova le classi che gestiscono la riproduzione vera e propria dei brani simulando i vari comandi play,pause…
Il terzo testa il music player nella sua interezza controllando la corretta riproduzione di una playlist composta da brani di formato differente.
Il quarto testa la funzionalità di loop controllando se a loop attivato una canzone che termina ricomincia da capo.
Infine il quinto test controlla il funzionamento della modalità shuffle.

3.2 Divisione dei compiti e metodologia di lavoro

Le varie parti del software sono state così suddivise:

Cevoli Alessandro:
. sviluppo della Gui, interazione con l'utente, gestione della playlist e della groove box a livello grafico.

Gabellini Matteo:
. sviluppo core della riproduzione, salvataggio audio e delle dovute strutture dati, sia per il music player che per la groovebox, gestione della playlist, del loop e dello shuffle.

Insieme è stato effettuato il design con la definizione delle interfacce che sarebbero servite poi per integrare le diverse componenti sviluppate singolarmente(vedere figura 2.1.1).
Successivamente sono state sviluppate le classi che rappresentano i dati gestiti dalla groovebox all'interno del software.

Per integrare le parti di codice singolarmente sviluppate si è cercato di rendere le componenti il meno dipendenti possibile l'una dall'altra fin dalla definizione dell'architettura. Precisamente in tutte le parti di codice in cui una componente (es: view) interpella un'altra (es: controller), lo fa attraverso le relative interfacce senza preoccuparsi del tipo concreto dell'istanza.
Ciò ha permesso in un primo momento di sviluppare le parti autonomamente e parallelamente, mentre in fase di refactoring di non avere grossi problemi tipici

delle dipendenze fra codici (es: il refactoring di una delle parti del MVC non ha compromesso le altre).

Considerando lo sviluppo del progetto in parallelo il DVCS è risultato indispensabile e ci ha permesso di mantenere sempre una copia stabile del software in luogo sicuro e reperibile ad entrambi i membri del gruppo. Inoltre ha assicurato un backup in caso di grossi "disastri" attraverso la funzione di rollback.
Durante lo sviluppo le modifiche effettuate singolarmente dai membri del team hanno comportato tempi di implementazione differenti e, dato lo sviluppo in parallelo, la funzione di merge ha permesso di sincronizzare le parti di codice sviluppate.

3.3 Note di sviluppo

In diverse fasi dello sviluppo, come di norma, ci si è trovati a scegliere tra diverse possibili soluzioni implementative per le diverse parti in particolare:

Durante la modellazione del controller è sembrato più consono inserire in esso la riproduzione concreta della musica invece di inserirla nel model perché non rappresentava una modellazione di dati ma una parte core attiva che utilizza i dati contenuti nel model.
Altra ragione che ha comportato tale scelta è che le classi che riproducono l'audio utilizzano librerie (fornite con il JRE) che interagiscono con aspetti del sistema operativo (l'uso del comparto audio).

In fase di modellazione delle feature si è valutato se ricorrere al pattern decorator oppure al pattern chain of responsability. Alla fine si è scelto(come mostrato nelle sezioni precedenti) il secondo pattern perché rendeva il concetto di aggiunta di feature ad una playlist più indipendente rispetto alla gestione core di quest'ultima e rendeva l'aggiunta di più funzioni contemporaneamente "più naturale".

Durante le prove di esecuzione sulle JVM dei diversi sistemi operativi abbiamo riscontrato un bug all'interno di quella per Machintosh OSX.
L'errore consiste in un'errata restituzione dei dati inerenti agli indici della tabella della groove box (vedere classe view.tables.GrooveBox) successivamente alla visualizzazione di un JOptionPane (es: tentativo di eliminazione di una canzone dalla playlist senza averla selezionata o avvio di una canzone a playlist vuota).
Dopo diverse analisi di debugging del codice abbiamo capito che l'errore è dato da AWT, il quale successivamente alla chiusura di un JOption Pane per un tempo indefinito, oppure fino a minimizzazione e riapertura della finestra principale, restituisce un valore "null" come posizione del mouse.
Tale errore non è stato riscontrato su piattaforme Windows e Linux (precisamente Windows 7, Linux Debian 8 e Linux BackBox 4.0).

L'idea del progetto nasce dalla visione degli esempi forniti da Oracle per l'uso della libreria per la gestione audio.

Capitolo 4

Commenti finali

Giunti al termine dello sviluppo del progetto ci possiamo ritenere soddisfatti del lavoro svolto, l'unica parte di cui non ci sentiamo pienamente contenti è la velocità di caricamento delle tracce wav.

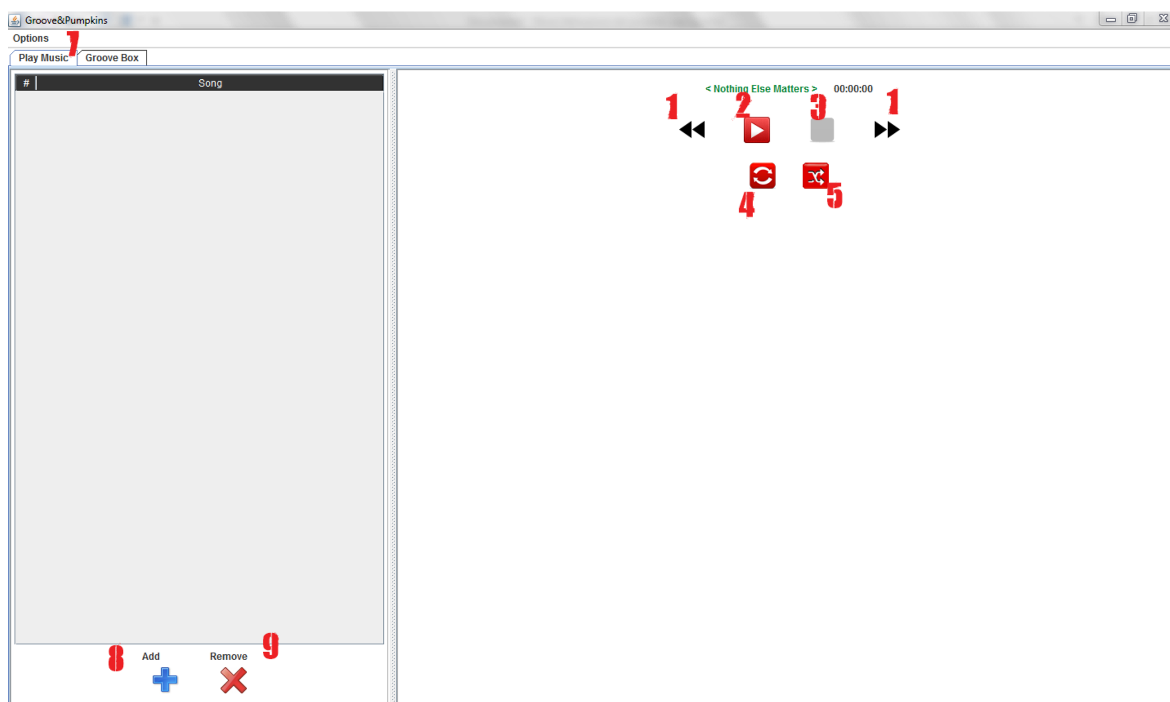
In futuro pensiamo di estendere il software attraverso:

- Miglioramento della velocità di caricamento delle tracce wav
- Keystroke da tastiera
- Registratore audio
- GrooveBox multilivello che permette di definire anche pattern melodici, oltre alla sola definizione di groove ritmici.

Appendice

Guida all’uso del software

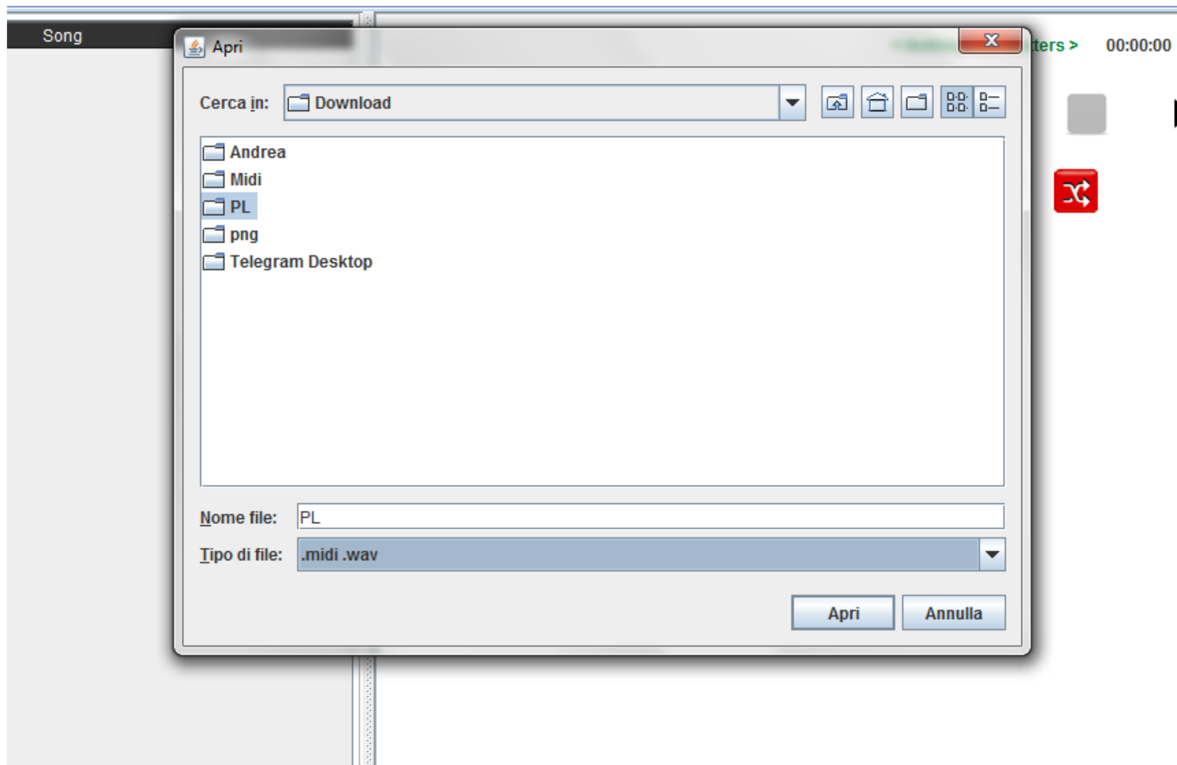
1. Interfacce del Music Player e relativa Playlist:



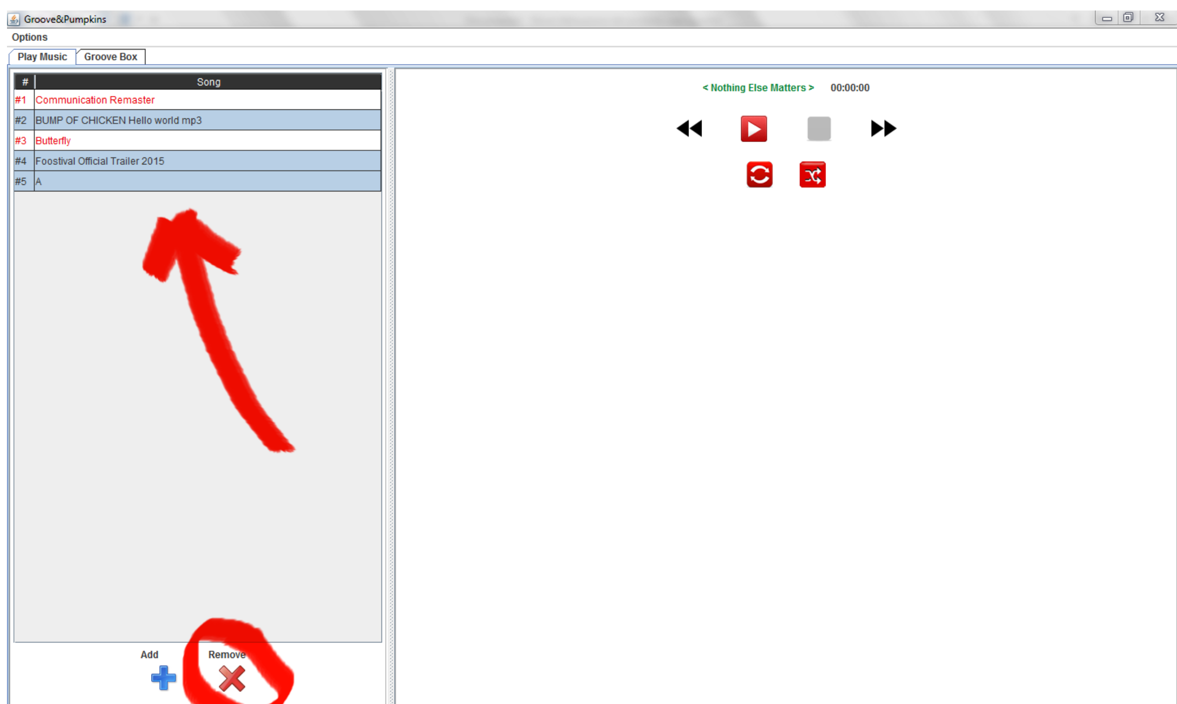
Tasti:

1. Pulsanti “Forward” (destra) e “Backward” (sinistra) per cambiare brano in riproduzione. In modalità normale scorre le canzoni sequenzialmente, se lo Shuffle è attivo il tasto Forward manderà in riproduzione una canzone causale, il tasto Backward pescherà la canzone precedentemente riprodotta. Se il Loop è attivo, manderà in Loop la canzone corrente. Di base lo Shuffle non termina se non per richiesta dell’utente, ripremendo il tasto Shuffle.
2. Pulsante “Play/Pause”, manda i riproduzione/pausa la canzone corrente. Per avviare una canzone a scelta, fare doppio click su di essa dalla playlist.
3. Pulsante Stop, ferma la riproduzione della canzone corrente corrente. La riproduzione verrà fermata anche navigando le tab dell’interfaccia ma non resetterà eventuali parametri impostati, playlist o pattern della groovebox.
4. Tasto “Loop”, manda in loop la canzone corrente.
5. Tasto “Shuffle”, permette la riproduzione casuale dei brani caricati nella playlist.
6. (7) Tasto opzioni, permette di chiudere il programma o di visualizzare la licenza allegata.
7. (8) Pulsate di aggiunta delle canzoni, apre il file manager da cui scegliere le canzoni da caricare. Il lettore ammette solo canzoni di tipo .midi o .wav. dal

file manager si possono caricare direttamente cartelle di brani selezionando la cartella e premendo invio (non caricherà le sotto cartelle per un motivo di sicurezza). Oppure selezionare uno o più brani selezionandoli normalmente.

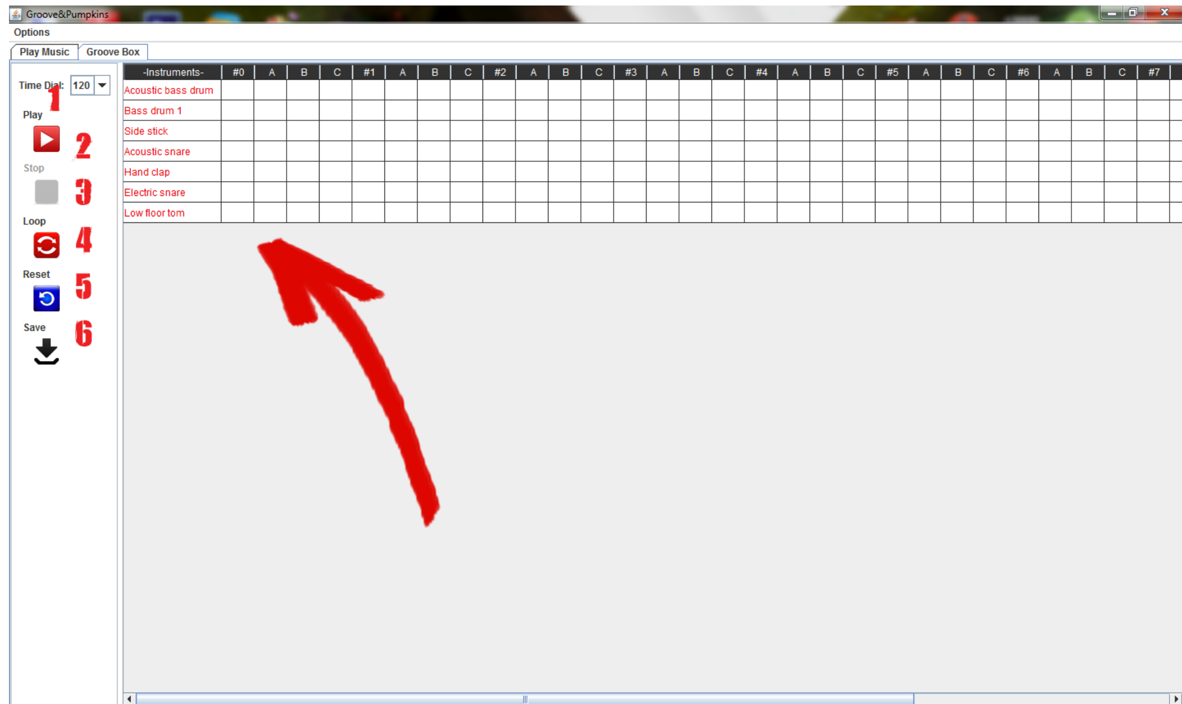


8. (9) Tasto “Remove” consente la rimozione di una o più canzoni dalla playlist tra quelle caricate.



Facendo click destro su una canzone della playlist compariranno le opzioni “Add” e “Remove”, per le funzioni vedi i punti 7 e 8.

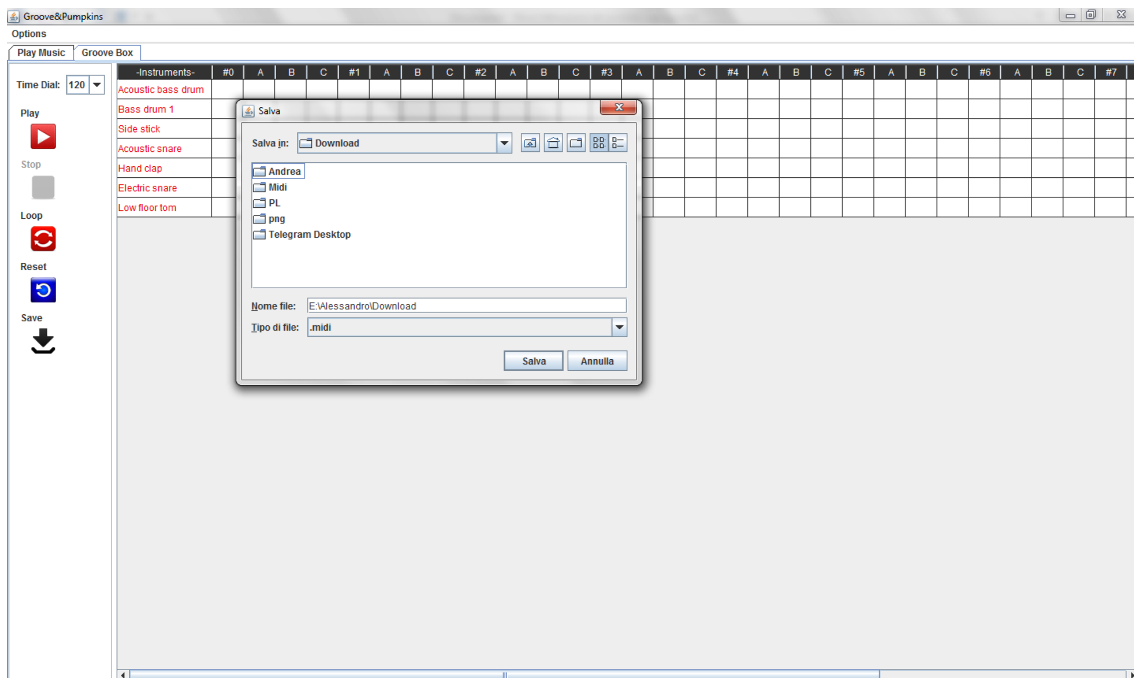
2. GrooveBox:



La Groovebox permette di creare pattern sonori clickando sopra i quadratini bianchi visualizzati. Ogni riga rappresenta un suono predefinito dal sistema, ogni colonna corrisponde a circa un 1/8 di secondo, le celle numerate segnano il mezzo di secondo.

1. La combo box “Time Dial” permette di scegliere la velocità (in beat per minutes) di riproduzione del pattern creato.
2. Il tasto “Play/Pause” permettono di avviare\pausare la riproduzione del pattern creato.
3. Il tasto stop ferma la riproduzione del pattern.
4. Il tasto “Loop” mette in ciclo il pattern creato durate la riproduzione.
5. Il tasto di “Reset” resetta la Groovebox a quella di default.
6. Il tasto “Save” permette di salvare i pattern creati in un file di tipo .midi nella cartella desiderata (quella di default è la home).

Facendo click destro su una cella della Groovebox verranno visualizzate le opzioni per invertire i colori della riga corrente o della colonna corrente.



Riferimenti

In primis la java doc:
https://docs.oracle.com/javase/8/docs/api/index.html

Esempi di uso della librerie per la gestione dell'audio forniti da Oracle:
http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html (Java SE Development Kit 8u45 Demos and Samples)

Studio del Pattern Chain of Responsability:
http://www.informatica.uniroma2.it/upload/2010/LIS/07-Pattern%20GOF.pdf
http://it.wikipedia.org/wiki/Chain-of-responsibility_pattern

Quali funzioni usare per effettuare il salvataggio di un file in formato midi:
http://www.mokabyte.it/2001/12/midi_3.htm

Per i test vengono letti dei brani da una directory interna al progetto ma fuori dalla directory bin per capire come fare si è guardato l'esempio d'uso della funzione "System.getResource("user.dir")" da questo forum:
http://stackoverflow.com/questions/11251289/how-to-read-a-properties-file-in-java-from-outside-the-class-folder

Contatti degli sviluppatori:

Alessandro Cevoli : alessandro.cevoli2@studio.unibo.it
Matteo Gabellini: matteo.gabellini2@studio.unibo.it