VirtualPiano

Giada Gibertoni, Meshua Galassi e Lorenzo Valentini

30 maggio 2016

Indice

1. Analisi

1.1 Idea

L'idea di questo progetto è nata osservando che recentemente i veri e propri strumenti musicali sono sempre più in disuso e vengono sostituiti da programmi o strumenti che simulano il loro funzionamento per motivi economici e di portabilità. L'obiettivo che ci siamo posti è quello di creare un'applicazione che può imitare strumenti musicali o creare suoni ed effetti partendo dai tasti di una tastiera.

1.2 Requisiti

Realizzazione di un'applicazione che simula una tastiera musicale multimediale.
La tastiera è formata da 2 ottave visibili con la possibilità di cambiare tonalità fino a 7 ottave totali.
È inoltre possibile registrare la traccia appena prodotta e riprodurla.
È prevista la selezione dello strumento che si intende suonare attraverso una combo box. La lista degli strumenti viene ricavata dal sintetizzatore di default presente nel JDK.
È possibile scegliere tra 5 diversi skin della tastiera che possono essere integrati con skin aggiuntivi in un secondo momento.
Nell'interfaccia è presente anche un menù help che mostra le note relative ad ogni tasto e i tasti della tastiera associati.

1.3 Miglioramenti futuri

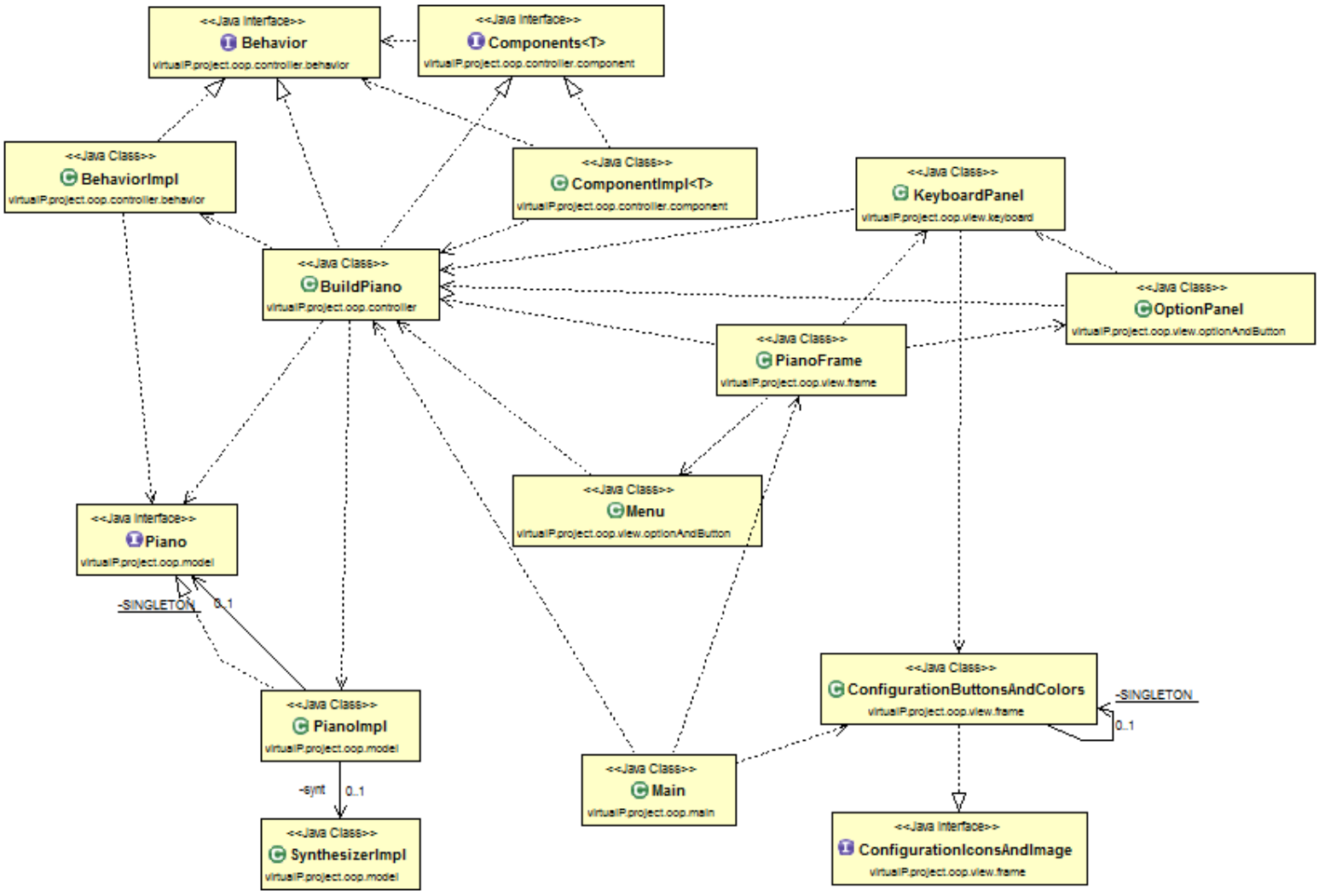
In futuro si cercherà di inserire la possibilità di suonare accordi o suonare una nota per una durata non stabilita a priori dall'applicazione.
Si cercherà, inoltre, di migliorare l'opzione di riproduzione e salvataggio delle tracce registrate.

Si potrebbe migliorare il collegamento dei tasti della tastiera ai *button* del piano utilizzando la classe KeyListener e ciò che ne concerne anziché modificare il *KeyboardFocusManager* di default.

2. Design

2.1 Architettura complessiva

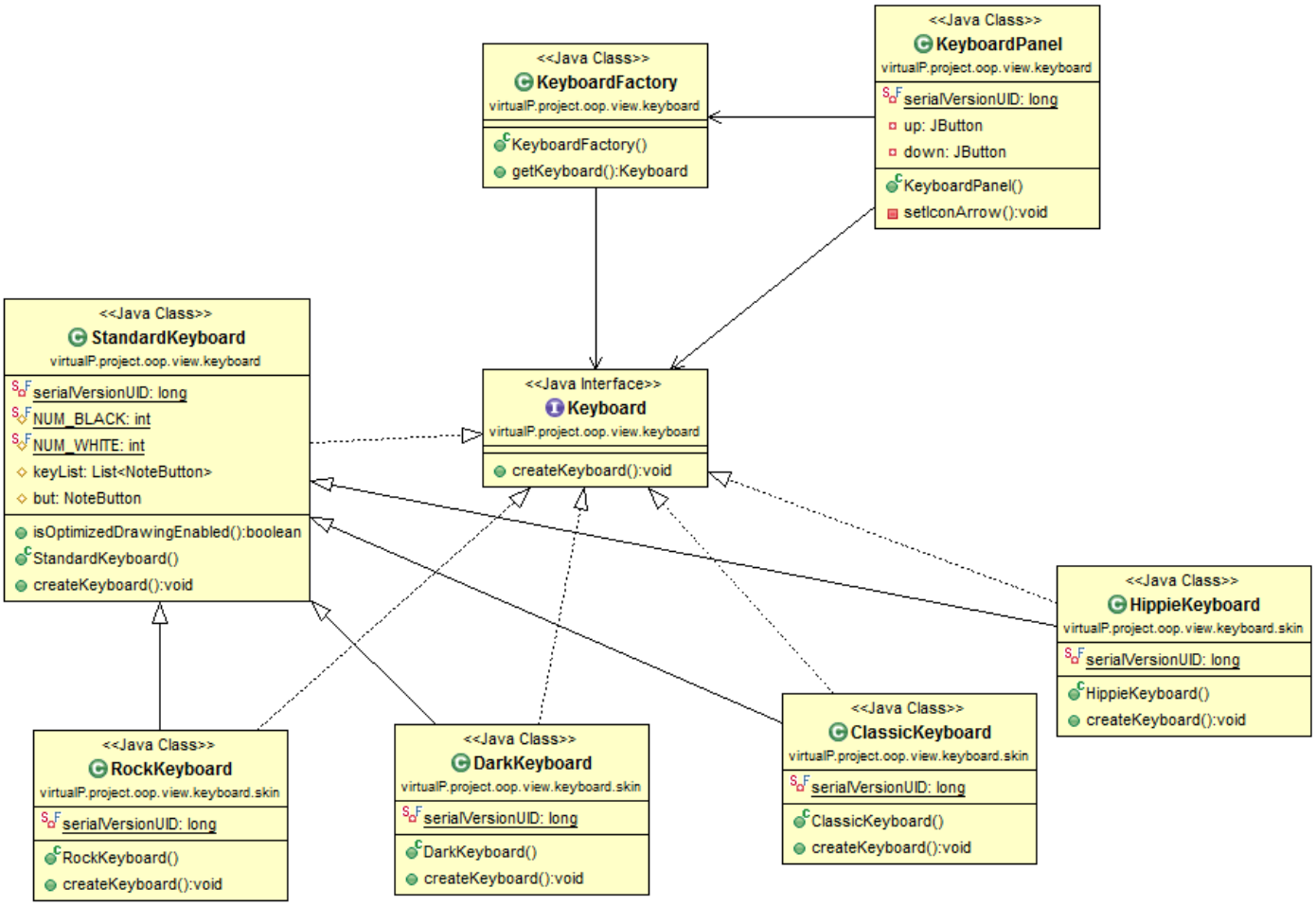
In questa applicazione abbiamo usato il pattern *Model-View-Controller*.
La comunicazione tra gli elementi della View e quelli del Model avviene attraverso il lavoro del Controller. Qui di seguito un UML generale con le classi principali dell'applicazione:



2.2 Pattern Utilizzati

2.2.1 View

Factory Method

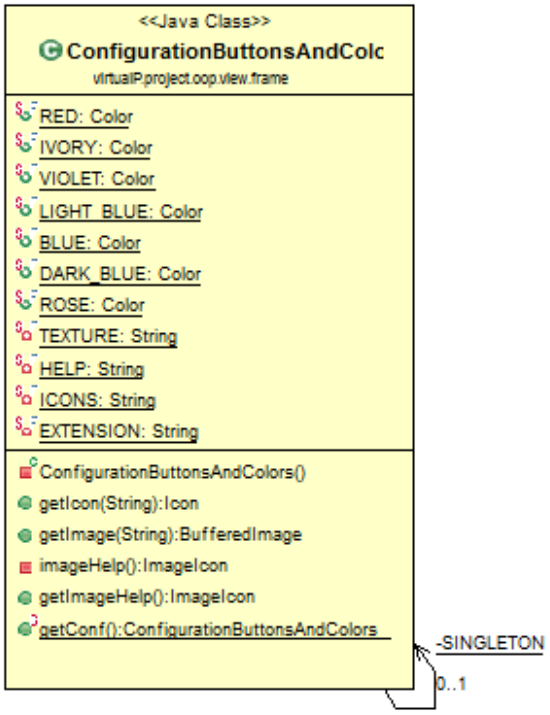


Si è utilizzato il pattern *Factory Method* per la costruzione della tastiera e le sue skin predefinite. Si è scelto di utilizzare questo pattern per rendere più intuitivo e compatto il codice e in oltre per rendere più indipendente l'aggiunta di skin in futuro.

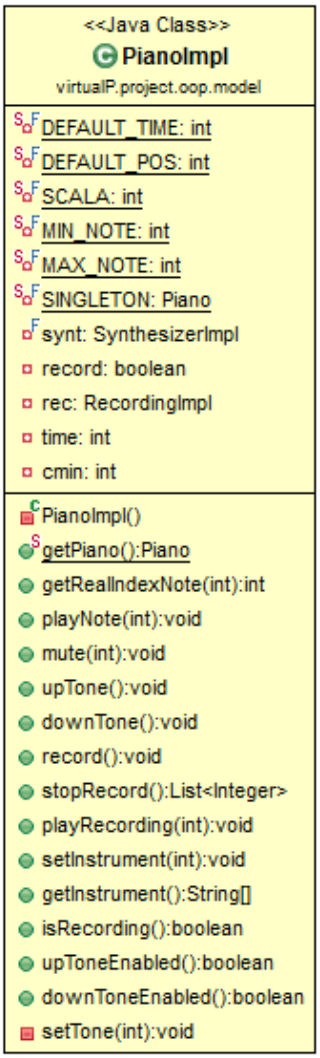
Singleton

Si è scelto di utilizzare il pattern *Singleton* per la classe che si occupa della configurazione iniziale del *look and feel* dell'applicazione, del reperimento delle varie immagini e icone e di fornire i colori alle varie classi che li richiedono.

Grazie a questo pattern è possibile creare una sola volta la classe *ConfigurationButtonsAndColors* che verrà poi utilizzata staticamente da tutte le classi che ne hanno bisogno.



2.2.2 Model



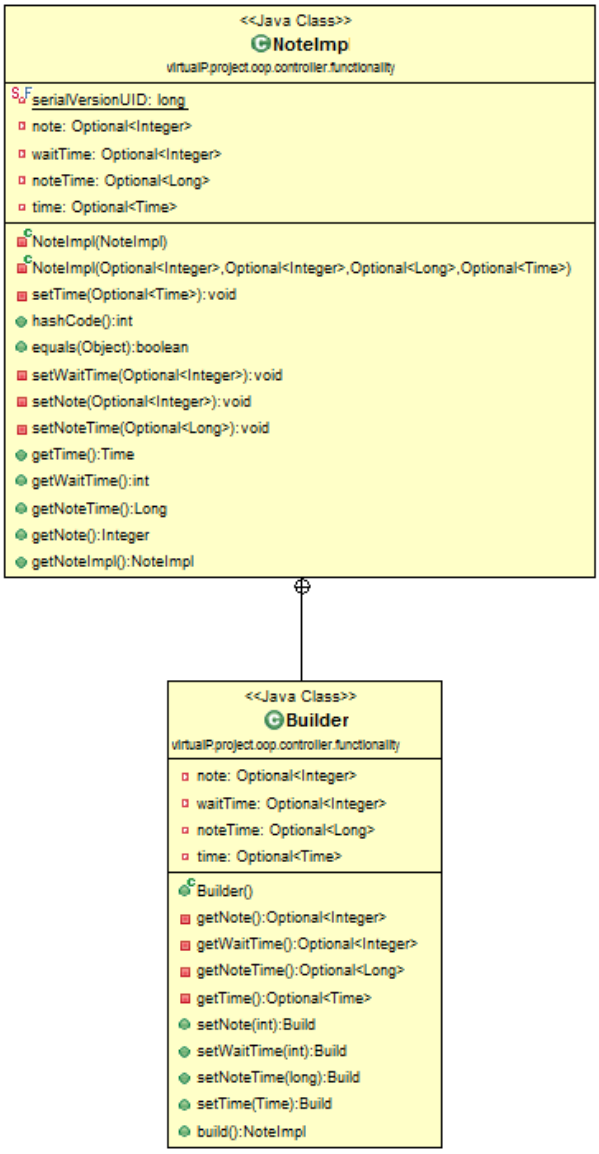
Singleton

È stato usato il pattern *Singleton* per la classe *PianoImpl*.

In questo modo è possibile creare un solo oggetto di tipo Piano che verrà utilizzato dal controller.

2.2.3 Controller

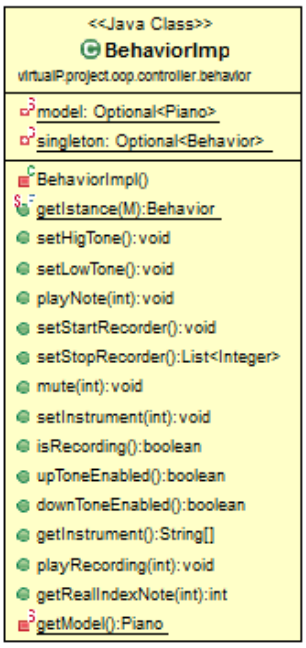
Builder



È stato utilizzato il pattern Builder nella classe *NoteImpl* per rendere la creazione dell'oggetto più flessibile e garantirne l'immutabilità dopo la creazione.

Un oggetto della classe *NoteImpl* dopo la sua creazione diventa immutabile poiché i suoi campi sono accessibili solo in lettura in modo da rendere l'oggetto "*nota*" non modificabile dall'esterno.

Singleton



BehaviorImpl:

La classe *BehaviorImpl* implementa l'interfaccia *Behavior* che definisce il comportamento di un oggetto "*piano*".

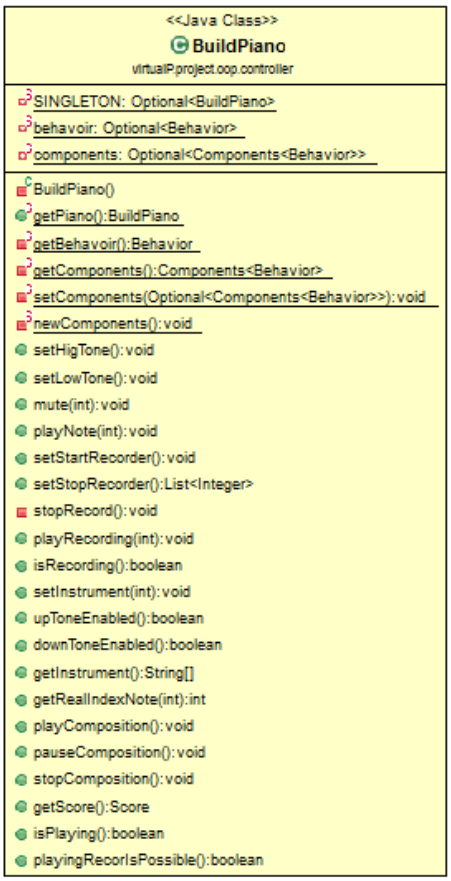
In questa classe si è scelto di utilizzare il pattern *Singleton* per far sì che all' interno dell'applicativo esista al più un'istanza della classe *BehaviorImpl*. Inoltre, questa classe fin da subito è stata considerata come una entità che potrà essere modificata sia con altre implementazioni della classe *behavior,* che inizializzata con una nuova istanza di un oggetto che implementa l'interfaccia *Piano*.

È stato fatto un Overloading del metodo statico *getIstance()* per ottenere un oggetto di questa classe secondo due versioni differenti. La prima versione prende come argomento un oggetto che implementa l'interfaccia *Piano*; la seconda non ha argomenti poiché setta automaticamente il campo con l'oggetto di una classe che implementa tale interfaccia.

BuildPiano:

Questa classe utilizza il *Singleton* poiché deve essere creata una sola volta all'avvio dell'applicazione per fornire il mezzo di comunicazione tra la View e il Model.

Questa classe si compone di un istanza della classe che descrive il comportamento del piano e un'istanza che descrive le funzionalità dei componenti della View.



3. Sviluppo

3.1 Testing

3.1.1 Test JUnit

Il testing è stato effettuato sulle seguenti classi:

- *TestBehavoir*: testa le funzionalità di base della classe che implementa il comportamento.
- *TestBuildPiano*: testa le funzionalità di base della classe BuildPiano.
- *TestComponents*: testa le funzionalità di base della classe ComponentImpl.
- *TestClass*: controlla una parte della registrazione.
- *TestNote*: controlla una parte della registrazione.
- *TestScore*: controlla una parte della registrazione.

3.1.2 Testing manuale

È stato effettuato un test manuale del funzionamento attraverso l'interfaccia grafica.

3.2 Divisione dei compiti

Meshua Galassi si è occupata di implementare le classi relative al Model quali il sintetizzatore che si occupa di ricavare i suoni e riprodurli e *PianoImpl* che si occupa, invece, dei vari controlli richiesti dalla View, di far avere alla View le informazioni che gli occorrono e di salvare una lista di note suonate.

Giada Gibertoni si è occupata di implementare la parte relativa alla View creando una GUI interattiva che incapsula i metodi forniti da controller.

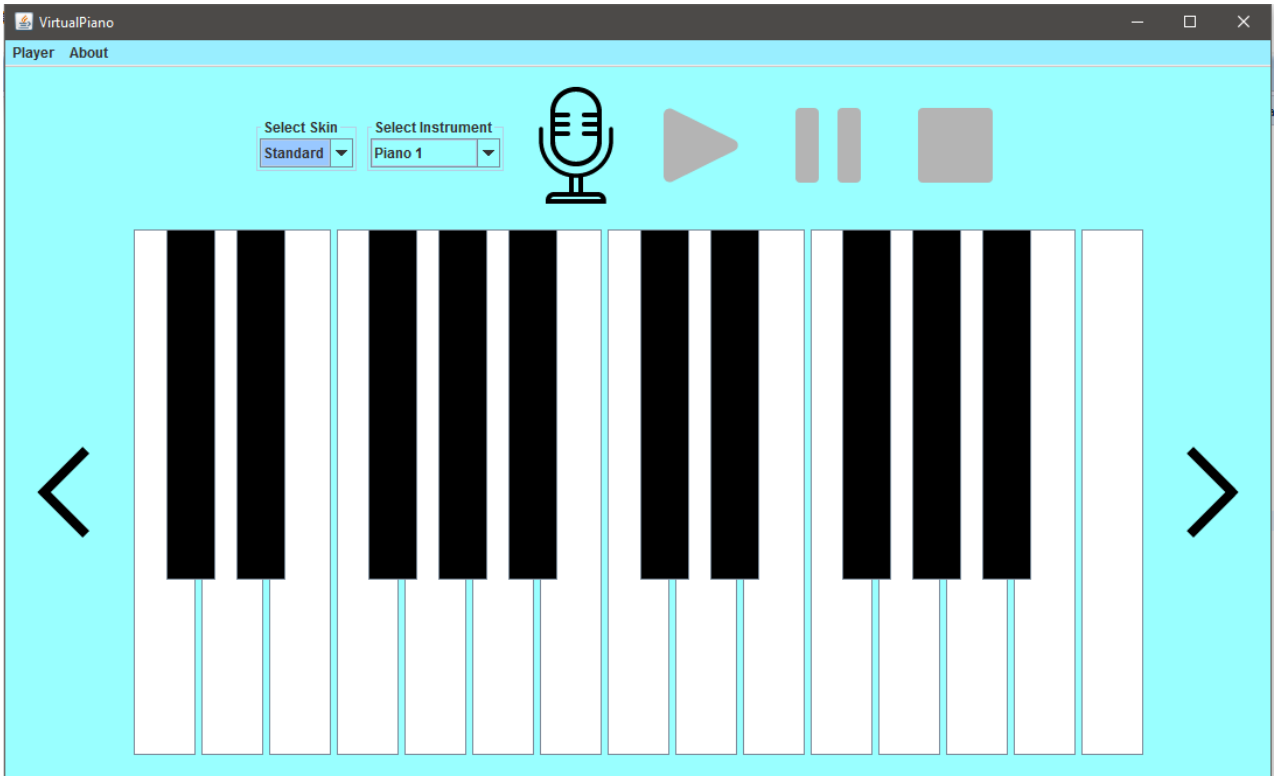
Lorenzo Valentini si è occupato di mettere in comunicazione View e Model e in particolare della gestione dei thread che consentono la riproduzione di suoni.

Le varie parti sono successivamente state integrate tra loro.

3.3 Difficoltà riscontrate

Abbiamo riscontrato difficoltà nella gestione dell'acquisizione di eventi da tastiera e l'abbiamo risolto in maniera non del tutto performante.

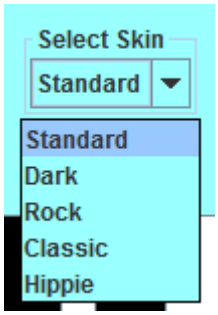
4. Guida Utente



All'avvio l'applicazione si presenta con una tastiera classica impostata sul piano forte. I tasti per la riproduzione audio sono disabilitati eccetto il tasto "*recorder*" per consentire l'avvio della registrazione.

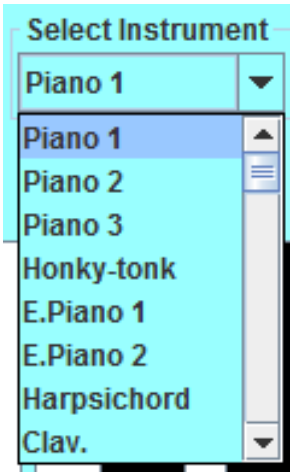
Premendo il tasto rappresentato da un microfono l'applicazione inizia a registrare le note suonate. L'icona relativa cambia colore e vengono disabilitati nuovamente i tasti per la riproduzione dell'audio finchè non si indica la fine della registrazione attraverso la medesima icona.

Una volta registrato un audio è possibile riprodurlo attraverso i tasti "*play*", "*pause*", "*stop*".



All'avvio di una nuova registrazione, quella precedente andrà perduta.

Attraverso il menù a tendina "*Select Skin*" è possibile cambiare lo skin di visualizzazione della tastiera con diversi temi.



Mediante il menù "*Select Instrument*", invece, è possibile selezionare uno dei diversi strumenti disponibili nel sintetizzatore.

Aprendo il menù *About* > *Help* è possibile avere una visione della tastiera con indicati per ogni tasto il nome della nota e la lettera della tastiera corrispondente che permette di suonarlo.