Titolo della tesi: **Sviluppo di un software per la gestione delle attività di una scuola di musica**Studente: Silvia Manca  
Matricola: 181421  
Corso di laurea: Ingegneria Gestionale (L8)

Sommario

[Struttura della prova finale 2](#_Toc446072016)

[Contesto operativo 3](#_Toc446072017)

[Descrizione del data-set 4](#_Toc446072018)

[Struttura dell’architettura 6](#_Toc446072019)

[Principali algoritmi utilizzati 8](#_Toc446072020)

[Conclusioni 11](#_Toc446072021)

# Indice delle figure

[Figura 1 Diagramma ER 5](#_Toc446072424)

[Figura 2 Struttura dei package 6](#_Toc446072425)

[Figura 3 Diagramma delle classi 7](#_Toc446072426)

[Figura 4 Messaggio di errore del metodo doCercaStudente() 9](#_Toc446072427)

[Figura 5 Risultato del metodo elencaLiv1() 9](#_Toc446072428)

[Figura 6 Azionamento del DatePicker 10](#_Toc446072429)

[Figura 7 Risultato del metodo doScegliBasso(MouseEvent event) 10](#_Toc446072430)

# Struttura della prova finale

|  |  |
| --- | --- |
| Studente proponente (matricola, cognome, nome) | 181421 - Manca Silvia |
| Titolo della prova finale | Sviluppo di un software per la gestione delle attività di una scuola di musica |
| Descrizione del problema proposto | Una scuola di musica in espansione, con circa 20  studenti attualmente, vorrebbe digitalizzare la gestione degli allievi e delle attività svolte.  Il software sarebbe utilizzato da una sola persona. |
| Descrizione della rilevanza gestionale del problema | Gestione degli studenti e delle attività |
| Descrizione dei data-set per la valutazione | Il software deve trattare diversi tipi di informazione quindi diversi data-set.  - Nominativi: informazioni anagrafiche sugli studenti  - Livell e Percorsi formativi: diverse fasce di competenza in cui inserire gli studenti  - Attività da svolgere: esercizi specifici da fare in ogni lezione in base al livello e del percorso formativo  - Lezioni: informazioni su ogni singola lezione di ogni studente (con le attività previste e poi effettivamente svolte). |
| Descrizione preliminare degli algoritmi coinvolti | - Ricerca e modifica di record nel database  - Visualizzazione modalità calendario |
| Descrizione preliminare delle funzionalità previste per l’applicazione software | - Inserimento dei nominativi (con tutte le informazioni anagrafiche), visualizzazione del loro dossier (lezioni effettuate e attività svolte in ognuna di esse)  - Ricerca di un nominativo - Inserimento di “percorsi formativi” preimpostati per i diversi livelli di studente.  Ogni percorso prevede delle attività specifiche da fare in ogni lezione. - Visualizzazione in modalità agenda delle lezioni giornaliere: per ogni record dell’agenda è possibile visualizzare alcune informazioni sullo studente e sulle attività che è previsto svolga per quella lezione, a seconda del percorso formativo e del livello in cui è inserito. |

# Contesto operativo

L’idea della creazione di questa applicazione JavaFX nasce dalla necessità di una scuola di musica, con sede in provincia di Lecce, di digitalizzare l’anagrafica degli studenti e la documentazione riguardo le attività svolte.

La scuola di musica in questione è specializzata nell’insegnamento della batteria ma offre altri 3 corsi musicali: basso, canto, chitarra.  
Considerando la maggiore richiesta per il corso di batteria sono state fatte delle richieste generali ed altre specifiche per il solo corso di batteria.

Le funzioni principali richieste sono:  
- Inserimento di uno studente  
- Visualizzazione degli studenti, divisi per corso  
- Visualizzazione dell’anagrafica dello studente

Le funzioni aggiuntive sono:  
- Visualizzazione dei percorsi formativi sullo studio della batteria  
- Inserimento di una nuova lezione del singolo studente  
- Visualizzazione del dossier dello studente, con lo storico delle sue lezioni  
- Visualizzazione in “modalità agenda” delle lezioni in programma e di quelle passate

Per il solo corso di batteria sono stati definiti 3 livelli di insegnamento distinti nei quali inserire gli studenti al momento dell’iscrizione a seconda della preparazione iniziale.  
Ogni livello quindi descrive in maniera dettagliata le attività che lo studente svolgerà durante il suo percorso formativo.

# Descrizione del data-set

Il dataset è composto da 8 tabelle, delle quali:

* 4 contengono i dati degli studenti, una per ogni corso di musica
* 3 contengono i dati dei percorsi formativi, una per ogni livello
* 1 contiene i dati delle lezioni

Le 4 tabelle rappresentanti gli studenti prendono il nome del corso di musica, ovvero *bassisti*, *batteristi*, *cantanti*, *chitarristi* e contengono una serie di dati anagrafici e descrittivi dello studente, ovvero:

* *`stud\_ID`* int: chiave primaria e identificativo univoco dello studente
* `*stud\_NOME*` varchar: nome dello studente
* `*stud\_COGNOME*` varchar: cognome dello studente
* `*stud\_GENERE*` bit(1): genere dello studente, 0 se di genere maschile e 1 se di genere femminile
* `*stud\_DATAN*` date: data di nascita dello studente
* `*stud\_INDIRIZZO*` varchar: indirizzo di residenza dello studente
* `*stud\_CITTA*` varchar: città di residenza dello studente
* `*stud\_CAP*` varchar: CAP di residenza dello studente
* `*stud\_CELL1*` varchar: primo numero di cellulare
* `*stud\_PROPR1*` varchar: proprietario del primo numero di cellulare (es. “personale”, “genitore”)
* `*stud\_CELL2*` varchar: secondo numero di cellulare,
* `*stud\_PROPR2*` varchar: proprietario del secondo numero di cellulare (es. “personale”, “genitore”)
* `*stud\_STRUMENTO*` varchar: corso musicale a cui si è iscritti nella scuola
* `*stud\_LIVELLO*` int: livello musicale in cui si è inseriti (“1”, “2”, “3” per il corso di batteria, “1” per tutti gli altri)

Le 3 tabelle rappresentanti i livelli del corso di batteria ovvero *livello\_1*, *livello\_2, livello\_3* contengono le informazioni riguardo le attività da svolgere con gli studenti e sono costituite da:

* `*liv\_NUM*` int: chiave primaria insieme a `*liv\_ID*` che identifica il livello
* `*liv\_ID*` int: chiave primaria insieme a `*liv\_NUM*` che identifica l’attività del livello
* `*liv\_ARGOMENTO*` text: descrizione dell’attività

La tabella *lezioni* contiene i dati riguardanti tutte le lezioni degli studenti del corso di batteria ed è costituita da:

* `*lez\_IDStud*` int: chiave esterna, rappresentante lo studente
* `*lez\_NumLiv*` int: chiave esterna, rappresentante il livello dello studente
* `*lez\_IDLiv*` int: chiave esterna che identifica l’attività svolta
* `*lez\_Data*` date: data della lezione
* `*lez\_Orario*` varchar: orario della lezione
* `*lez\_Commento*` longtext: nota aggiuntiva (es. “portare spartiti”)
* `*lez\_Valutazione*` longtext: valutazione della lezione (non usata in questa versione dell’applicazione)

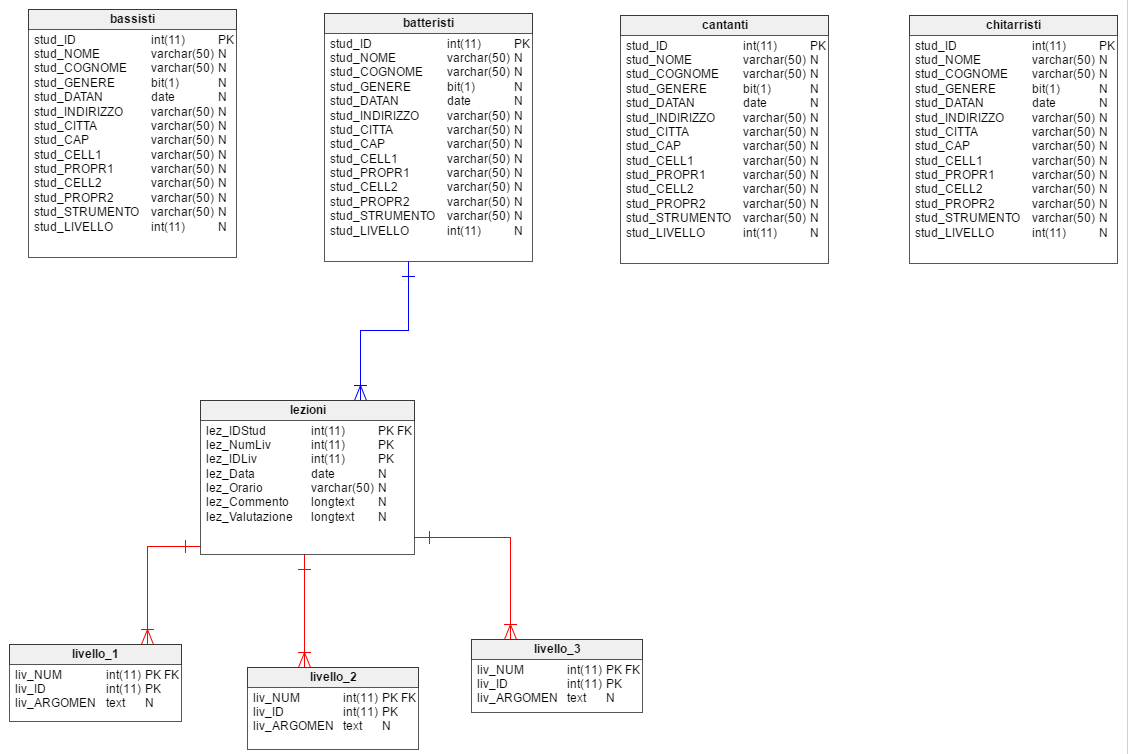


Figura 1 Diagramma ER

# Struttura dell’architettura

L’applicazione è stata sviluppata secondo il pattern *MVC (Model-View-Controller).*

Il progetto di lavoro JavaFX si estende su 5 package, come riportato in Figura 2 Struttura dei packageFigura 2.  
Il package *icons* contiene le icone per ogni *tab*, inserite tra i tag <image> </image> nel file *screentab.fxml*.   
Il package *software* contiene il *Main*, la classe *Model* e i *Controller*.  
Il package *bean* contiene le classi che rappresentano lo studente e la lezione.  
Il package *dao* contiene le classi per l’accesso ai dati nel database.  
Il package *gui* contiene i file di interfaccia *.fxml* e i fogli di stile.

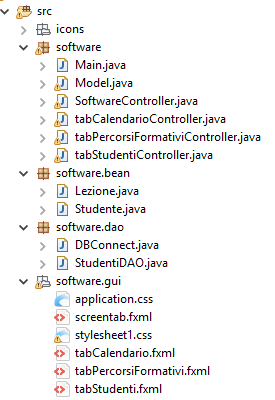


Figura 2 Struttura dei package

Per quanto riguarda la parte grafica il *Pane di root* utilizzato è un *BorderPane*, nella cui parte centrale è stato inserito un *TabPane* che permette una divisione ordinata dei diversi ambiti dell'applicazione.

Considerata l’elevata quantità di elementi da inserire, per una maggiore chiarezza ad ogni tab è stato assegnato un proprio file *.fxml*, inserito poi nel file principale, ovvero quello chiamato dal *loader* in fase di avvio dell’applicazione, attraverso la funzione fx:*include*.

<fx:include fx:id=*"tabStudenti"* source=*"tabStudenti.fxml"* />

Attraverso il metodo ControllerFactory()della classe javaFX.fxml.FXMLLoader è stato assegnato un diverso Controller per ogni file *.fxml*, ma un'unica classe M*odel*.   
  
Callback<Class<?>, Object> controllerFactory = type -> {  
 **try** {  
 **for** (Constructor<?> c : type.getConstructors()) {  
 **if** (c.getParameterCount() == 1 && c.getParameterTypes()[0] == Model.**class**) {  
 **return** c.newInstance(model); } }  
 **return** type.newInstance(); } };

Quanto descritto è più facilmente comprensibile tramite il diagramma delle classi del progetto, riportato di seguito.

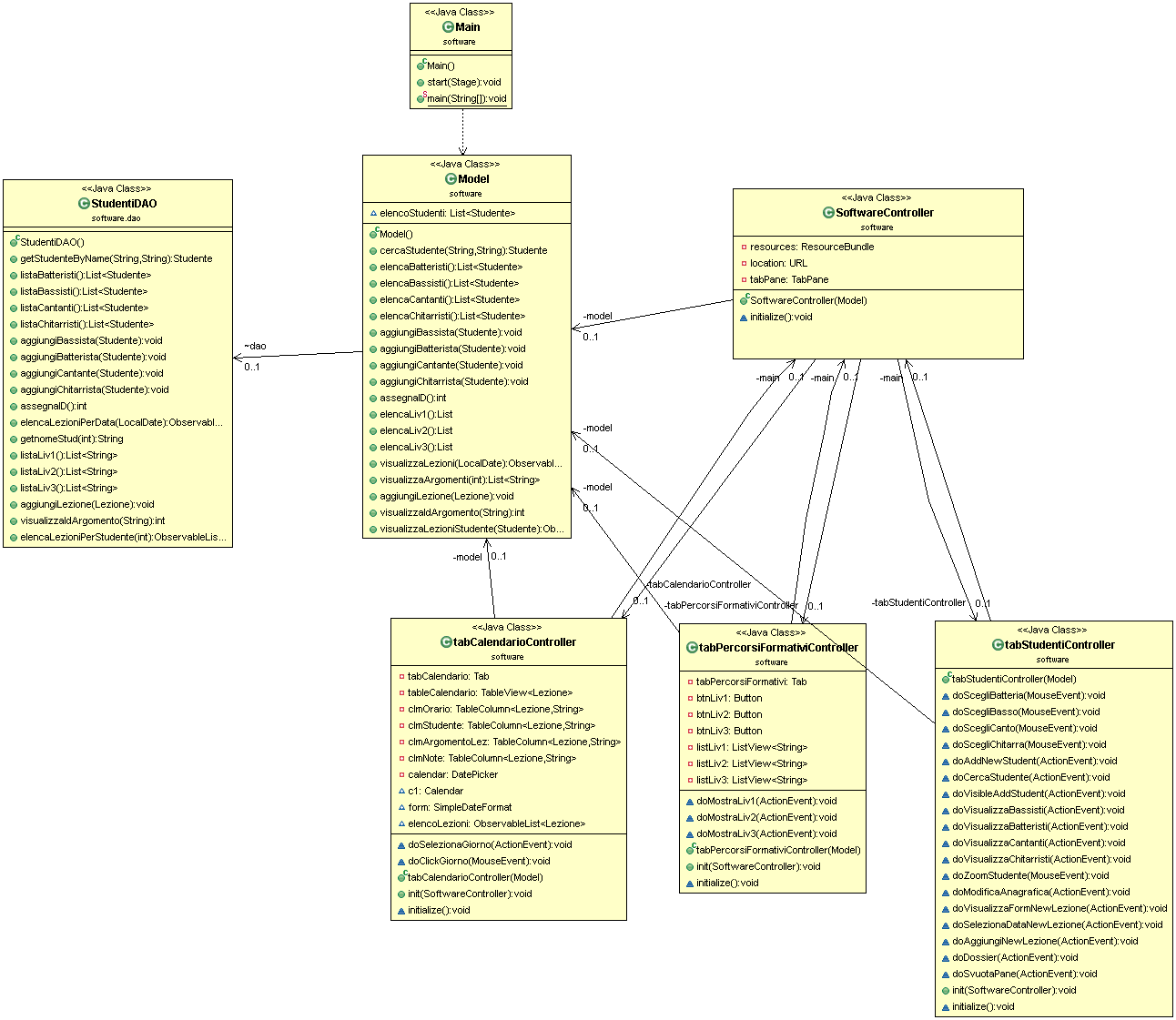


Figura 3 Diagramma delle classi

# Principali algoritmi utilizzati

I metodi implementati nell’applicazione si possono dividere in due tipologie: metodi per la **ricerca** e metodi per l**’inserimento** di record nel database.  
  
Tra quelli per la **ricerca** si distinguono:

* Metodi con il **passaggio di un parametro**, inserito dall’utente

(Di seguito viene riportato, in breve, il funzionamento di un metodo appartenente a questa tipologia)

Il metodo doCercaStudente()della classe tabStudentiController richiama il metodo cercaStudente(String nome,String cognome) della classe Model, passando come parametri di tipo String il nome e il cognome inseriti dall’utente nel form di ricerca.

Il Model a sua volta chiama la funzione cercaStudente(String nome,String cognome) della classe StudentiDAO.   
Quest’ultimo effettua una chiamata al database tramite una *query* indirizzata all’unione delle tabelle con l’anagrafica (quindi *bassist*i, *batteristi*, *cantanti,* *chitarristi*), passando tramite il PreparedStatement i parametri nome e cognome.  
I risultati ottenuti riempiono il *costruttore* dello *Studente*, passato poi al Model e a sua volta passato al tabStudentiController che attraverso i metodi get() delle variabili delle classi con l’anagrafica e i metodi set() degli elementi grafici come i *Text* visualizza graficamente il risultato cercato.  
Nel caso di ricerca non andata a buon fine, il metodo restituisce un messaggio di errore. (Figura 4)

* Metodi con un **elenco** di risultati  
  Sviluppati in maniera simile al metodo descritto precedentemente ma con la differenza del salvataggio dei record ottenuti dalla *query* in *List* passate poi al *Model* e a sua volta al *Controller* e visualizzate tramite *ListView* o *TableView<>* .

Un esempio di questa categoria è il metodo elencaLiv1() del Model richiamato nella classe tabPercorsiFormativiController per visualizzare in una *ListView* le attività da svolgere, come si vede in fig.2.

listLiv1.setItems(FXCollections.*observableList*(model.elencaLiv1()));

I metodi per l’**inserimento** di un record sono:

* doAddNewStudent()della classe tabStudentiController che inserisce un nuovo Studente in una delle tabelle dell’anagrafica, a seconda dello strumento selezionato nel form di aggiunta dello studente, come si vede nella Figura 5.
* doAggiungiNewLezione () della classe tabStudentiController che inserisce una nuova lezione nella tabella *lezioni*.

Altri metodi sono associati alla classe *ActionEvent* e vengono azionati dall’utente cliccando un bottone o scegliendo una data nel *DatePicker* come si vede in Figura 6.  
Metodi di questo tipo sono stati utilizzati maggiormente per impostare la visibilità di alcune parti grafiche come *VBox* contenenti elementi da visualizzare solo in determinati casi.

Nell’esempio riportato di seguito è reso visibile solo il *MenuButton* contenente i livelli del corso di basso, in seguito all’azione dell’utente di selezionare uno strumento piuttosto che un altro, come si vede in Figura 7.  
 **void** doScegliBasso(MouseEvent event) { menuLivelloBasso.setVisible(**true**); menuLivelloBatteria.setVisible(**false**); menuLivelloCanto.setVisible(**false**);   
 menuLivelloChitarra.setVisible(**false**);}



Figura 4 Messaggio di errore del metodo doCercaStudente()

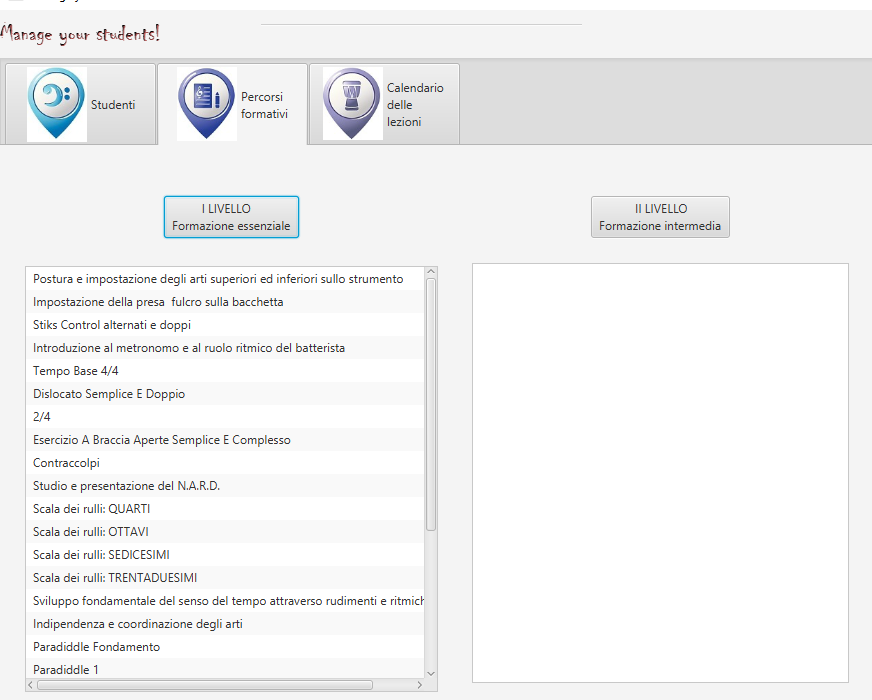


Figura 5 Risultato del metodo elencaLiv1()

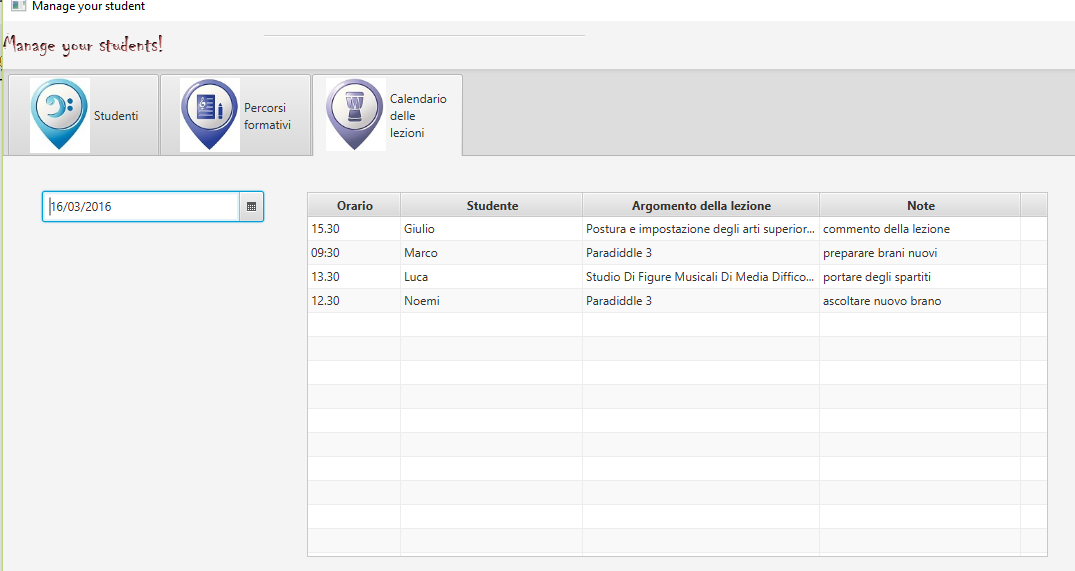


Figura 6 Azionamento del DatePicker

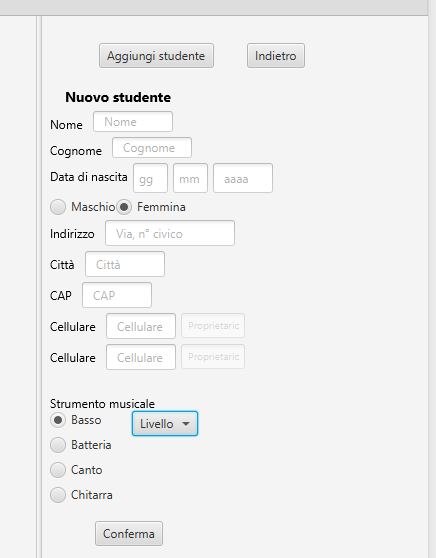


Figura 7 Risultato del metodo doScegliBasso(MouseEvent event)

# Conclusioni

L’applicazione è facilmente utilizzabile da qualsiasi utente medio o inesperto: la presenza del *Prompt Text* in tutti i *textField* e la comparsa di alcuni messaggi di errore nel caso di mancato completamento del form dovrebbero aiutare l’utilizzatore finale ad un approccio corretto dell’app.

Nonostante questi “accorgimenti” la prima implementazione da fare in una seconda versione dell’applicazione è l’incremento dei messaggi di errore, di consigli per il completamento dei form e di controlli nell’inserimento dei dati nel database.

La seconda implementazione da fare per rendere l’applicazione utilizzabile a tutti gli effetti è la funzione di modifica dell’anagrafica dello studente, per la quale è stato già inserito il bottone “Modifica anagrafica” nel tab *Studenti* e la modifica dei dati di una lezione inserita, prima o dopo che questa si sia svolta, aggiungendo per esempio una valutazione allo studente, aggiornare le attività effettivamente svolte o posticiparne la data.

L’applicazione si presta bene a futuri aggiornamenti come l’aggiunta di percorsi formativi per tutti i corsi musicali e l’inserimento di lezioni e della “modalità agenda” per gli altri corsi musicali

Al seguente link (e codice QR) è presente un breve video dimostrativo del funzionamento dell’applicazione.

<https://youtu.be/532TyJ12qjI>

