Mini-project Report

Course: IT3103

Topic: 10 – Electronic Piano

Class: OOLT.VN.20231

Group: 4

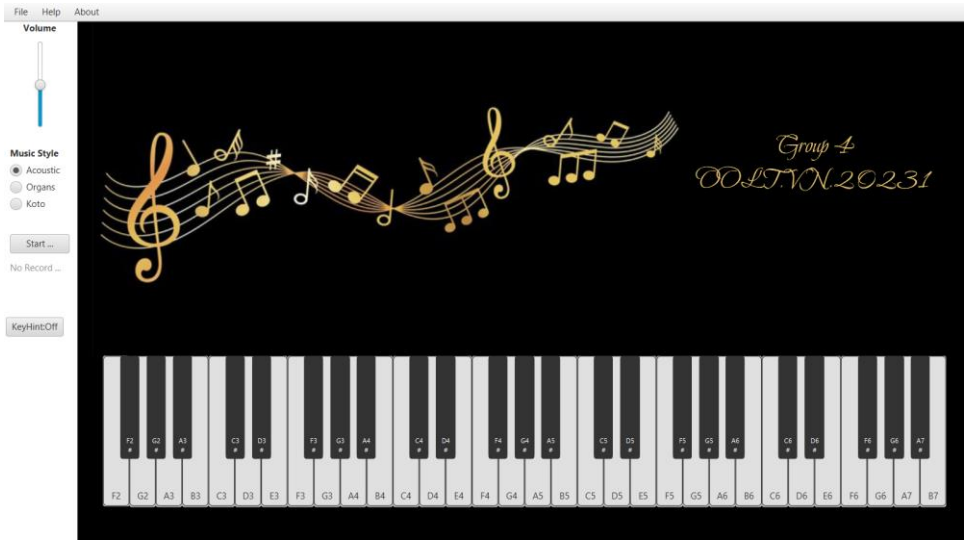
Members & Contribution

1. **Nguyễn Trực Cường - 20215005 (Leader)**
   - Design the application's main UI using FXML and CSS.
   - Implement JavaFX controllers for the application's components.
   - Design and implement model classes(Piano, PianoKey, Recorder,…).
   - Review team member codes.
   - Prepare texts and image resources for application.
   - Prepare report & presentation.
   - Integrate Apache Maven build tool into the project.
2. **Phạm Thành Công - 20194494**
   - Design the help menu.
   - Improve UX.
   - Provide suggestions for Class Diagram and Use Case Diagram.
3. **Đặng Minh Chức – 20215001**
   - Provide suggestions for Class Diagram and Use Case Diagram.
   - Prepare demo video.
4. **Khổng Lê Cường - 20215004**
   - Prepare piano sound data.
   - Research audio playback methods in Java.
   - Add new music styles to the application's setting.
   - Provide suggestions for Class Diagram and Use Case Diagram.

Project Description

This is a JavaFX application that simulates a 55-key electronic piano. Users can interact with the piano by clicking on the GUI piano keys or pressing specific keys on the keyboard. The application also comes with some extra features namely key hints toggling, play recording and replaying, volume adjustment, and different music style options. In addition, a help menu is also available with intuitive graphics.

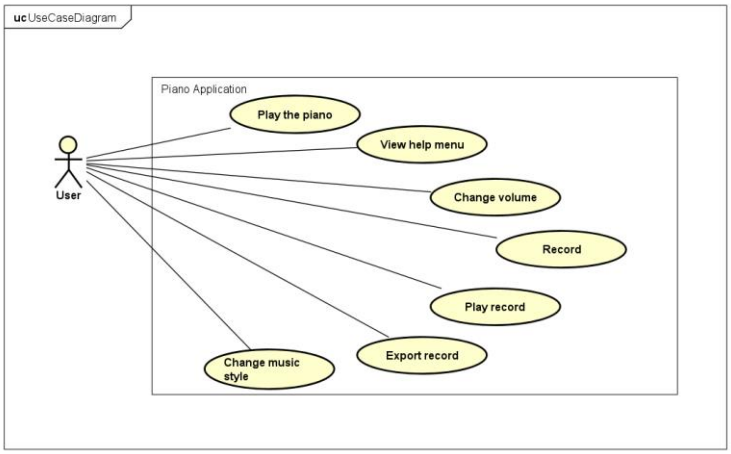
The underlying model classes of the application are built with object-oriented programming principles in mind, which will be further explained in the Design and Implementation details section.



Application User Interface

The 55-key Piano layout is inspired by Annex Softworks' Piano Player.

The application is designed based on the following Use Case Diagram:

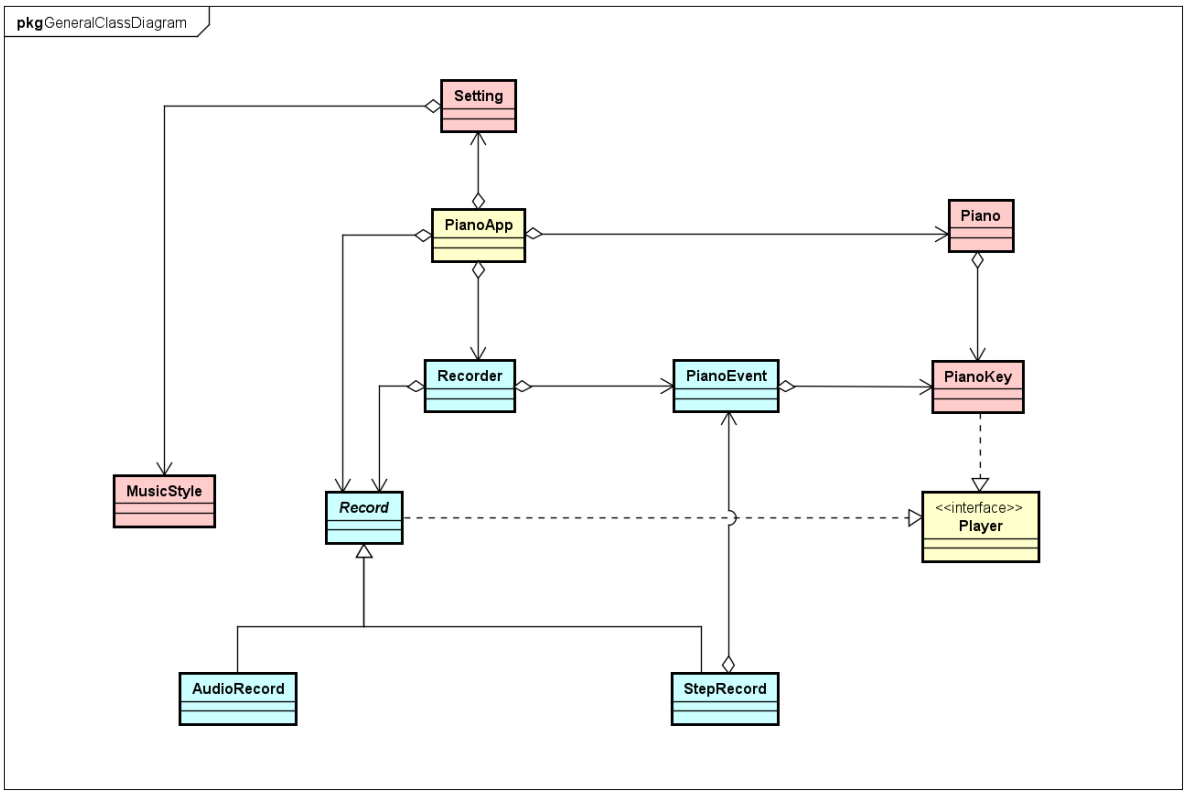


Use Case Diagram

As can be seen in the diagram, a user can do the following actions:

- **Play the piano** (clicking on the screen or pressing a key)
- **View help menu** (select ‘Help’ menu item on the menu bar)
- **Change volume** (drag the volume slider up or down)
- **Record** (press ‘Record’ button)
- **Play record** (import valid .csv file by selecting ‘Import Record…’ under the ‘File’ menu item on the menu bar then press ‘Play Record’ button)
- **Export record** (select export location after record then press ‘Export’)
- **Change music styles** (select the radio button with the corresponding music style names under ‘Music styles’ label)

Design and Implementation details



General Class Diagram

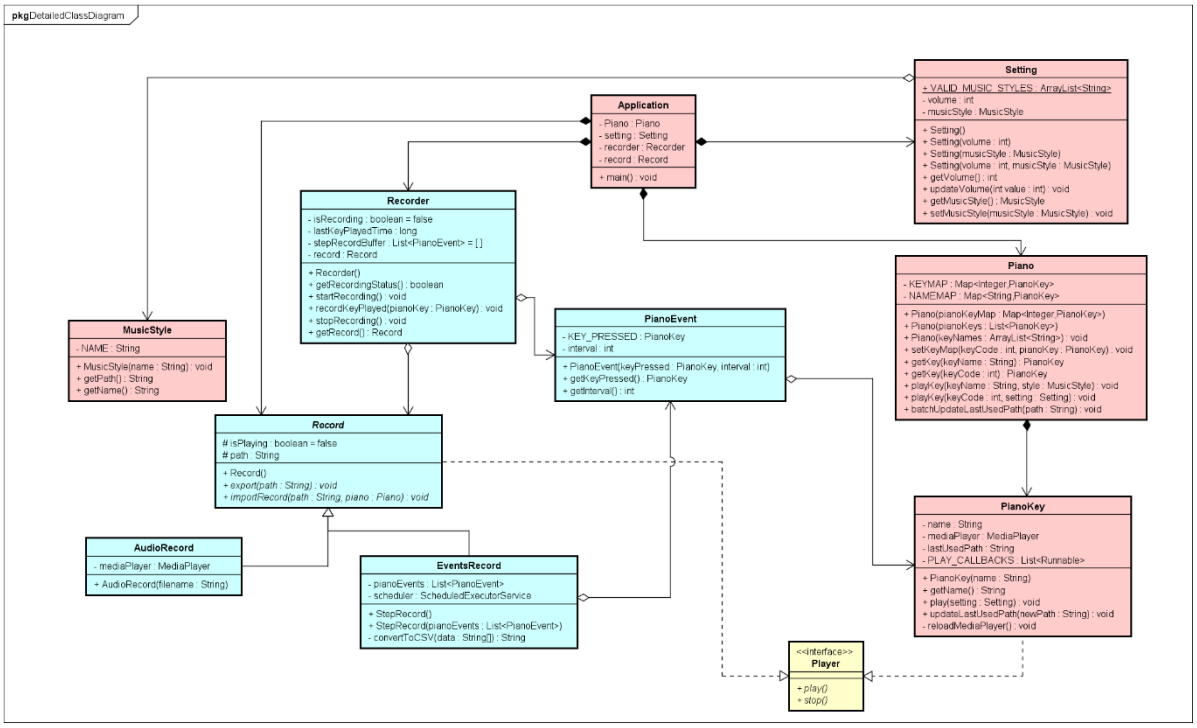
The general idea is that the application should be composed of 4 objects which are instances of 4 model classes. These objects are:

- A 'piano' object that represents the piano with many piano keys.
- A 'setting' object that controls how the piano sounds.
- A 'recorder' object that can record the user's action within the application.
- A 'record' object used to store imported play from external sources.

A piano contains many objects of **PianoKey** class. The setting object has **MusicStyle** as one of its attributes. The **Recorder** class can create and store a list of **PianoEvent** objects. An instance of **PianoEvent** is created whenever a **PianoKey** object is played AND there is a **Recorder** running. An instance of the **Recorder** class can also store a **Record**, which is an abstract class. **AudioRecord** and **StepRecord** are the two classes that inherit from **Record**. The difference between an **AudioRecord** and a **StepRecord** is defined as follows:

- **AudioRecord** represents raw audio data imported from a .wav file.
- **StepRecord** provides the step-by-step piano keypress. It should be able to store what key to press and how long before the key is pressed (the interval).

Both **Record** and **PianoKey** implement the **Player** interface.



Detailed Class Diagram

Aside from the obvious methods and attributes that serve the aforementioned general design, some methods and attributes are created for optimization purposes. For example, the **lastUsedPath** attribute in **PianoKey** class is used to reduce the number of times the play method has to re-query the music style path and apply it to the **MediaPlayer** (since the piano key pressing frequency is higher than that of changing music style for the average user). Basically, the **MediaPlayer** property only changes by **reloadMediaPlayer** which is called by **updateLastUsedPath** method. In the **Piano** class, the **batchUpdateLastUsedPath** can update all last used paths of all piano keys within it, which can be implemented in an application on music style change to improve performance. In **Piano** class, **NAMEMAP** is used to map the piano key name to the object itself, while **KEYMAP** uses a number (in this case the keystroke key number) to achieve the same.

In the main application, the piano, recorder, and setting communicate via messages, which follow Object-oriented principles. The composition relationship can be seen from the use of a **Piano** object as a 'container' of **PianoKey** objects. The creation of **PianoKey** objects is also performed within the **Piano** class. The implementation details of the **Record** class are also concealed within the export method. For example, in the children class **StepRecord**, the export method uses a private CSV converter method to perform the export action. This is to adhere to the Encapsulation principle. Also, the use of **Player** interface also allows Polymorphism in the sense that it was implemented by the abstract **Record** class and then passed down to its 2 children classes. When the play method is called from a record, it will run the play method of the respective children class. **StepRecord**'s play method should go through the

pianoEvents list and play each of the actual piano keys while **AudioRecord** would simply play the stored raw audio (.wav file).

The source code is currently missing the implementation of the creation of **AudioRecord** objects within the **Recorder** class due to Java language limitations in recording system audio. A workaround using a combination of Java Native Interface (JNI) and C/C++ might be possible, but we believe that it is out of scope for this project's purpose.