PROJECT REPORT

PROJECT SUMMARY

REPORT DATE	PROJECT NAME	PREPARED BY
15/7/2022	Electronic Piano	Nguyen Cong Dat – 20200137 Le Hong Duc - 20204874

1. MEMBER ASSIGNMENTS

NGUYEN CONG DAT 20200137	LE HONG DUC 20204874
MainScreen.java	VirtualPianoVer2Controller.java
HelpTextController.java	VirtualPianoVer2.java
MainScreenController.java	PianoNote.java
Help.fxml	Volume.java
Main.fxml	Record.java
Pianov2.fxml	Instrument.java
HelpSupporter.css	Piano.java
MainSupporter.css	Flute.java
PianoSupporter.css	Trumpet.java
help.txt	Violin.java
Do report	Guitar.java
Design presentation slides	BidirectionalMap.java
	FailedPatternException.java
	NullPianoNoteException.java
	Record demo video

2. MINI-PROJECT DESCRIPTION

2.1 PROJECT OVERVIEW

In this project, we get to build the first application, Electronic Piano, using Java and everything we have learned in the OOP course. We are not only implementing a GUI for the user to interact with piano keyboard, but we also add plenty of functional options just to bring the most realistic experiences.

While doing this project, we have to duel with a lot of difficulty in getting to know how a sound is created, the way every sound resonance with each other, as well as chord, pitch, timbre, rest, and many other attributes of music. Although we manage to overcome, there is still some restrictions we have to setup for the sake of the project:

1. The sustainability of sound is fixed, that is, it will extinguish soon after we release note.
2. Due to limited number of keys on keyboard, we only set 3 octaves at a time to play.

2.2 DESIGN REQUIREMENTS

- On the main menu: title of the application, piano GUI, help menu, quit
  - User can play the piano by interacting with GUI
  - Help menu shows the basic usage and aim of the program
  - Quit exits the program. Remember to ask for confirmation
- In the demonstration:
  - Keyboard: C (Do), D (Re), E (Mi), F (Fa), G (Sol), A (La), B (Xi), ...
  - A slider for increasing/decreasing volume
  - A record button to record the play
  - A button for changing music style (optional)
- We also add some more features to support the above requirements and add useful features for users to use:
  - An assistant-key to help user know which key refers to which note
  - A text field show every note we just play
  - A clear button to delete everything we just play
  - A replay button to replay what we have just played
  - A song button to play song we have recorded and default song of the app

2.3 USE-CASE DIAGRAM EXPLANATION

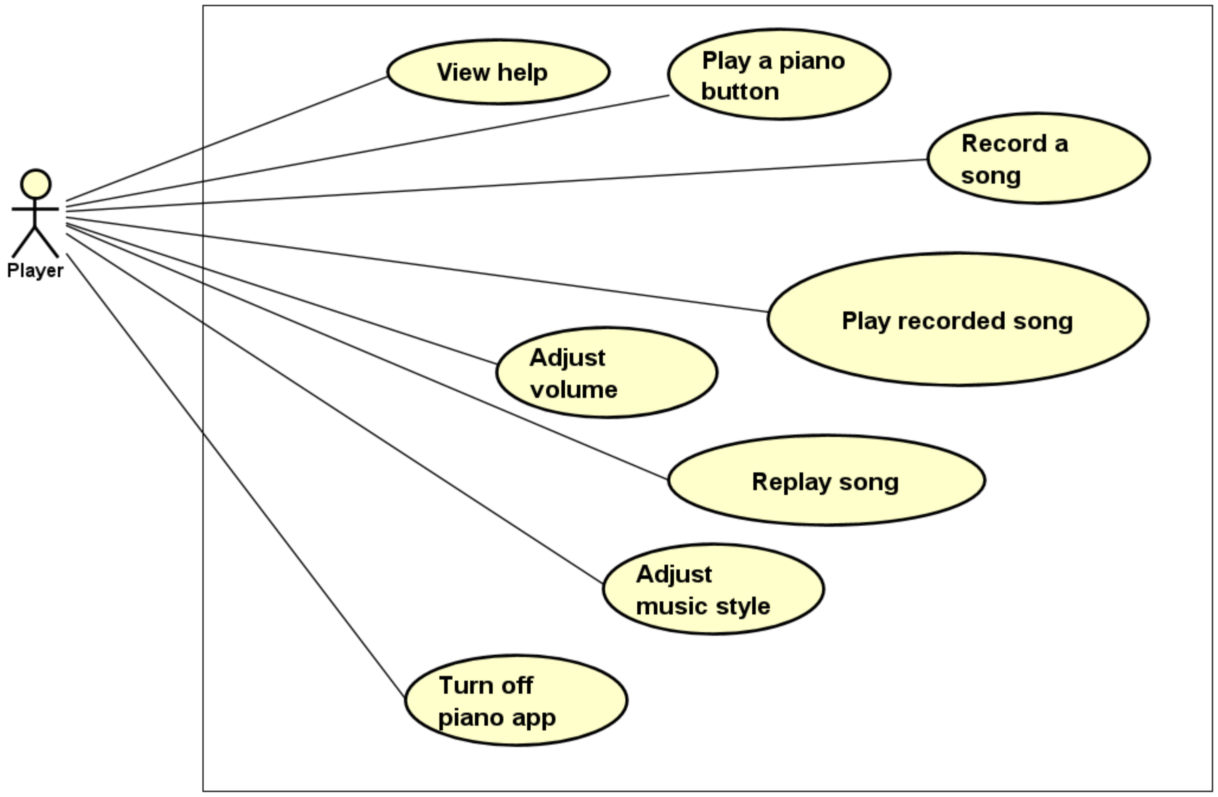

Figure 1. Use case diagram

- ❖ Based on the requirements we decide to develop eight use cases with 4 additional use cases. To demonstrate:
  - ➢ Firstly, the application allows user to click the piano note or press a key on keyboard, the app returns sound based on the specific note that user clicked or pressed
  - ➢ Notice the user if they press on a key that don't refer to any notes.
  - ➢ Every note user played will appear on a text field and it can replay song from those notes if user click the replay button.
  - ➢ When user click record button, the text field is made clear and now it starts to record the song. After finished, user can add a title for the song and save it.
  - ➢ User can choose either a default song or recorded song and play it, they can also remove a recorded song but not a default song.
  - ➢ Set the magnitude of volume
  - ➢ Set the style of music, volume and style can only be open 1 at a time
  - ➢ View help menu use case: shows the basic usage and aim of the program
  - ➢ - Turn off the app use case: First ask the user whether he wants to leave or continue staying in the app. Then based on his decision, close or remain current window

3. OOP DETAIL DESIGN EXPLANATION

3.1 GENERAL DESIGN

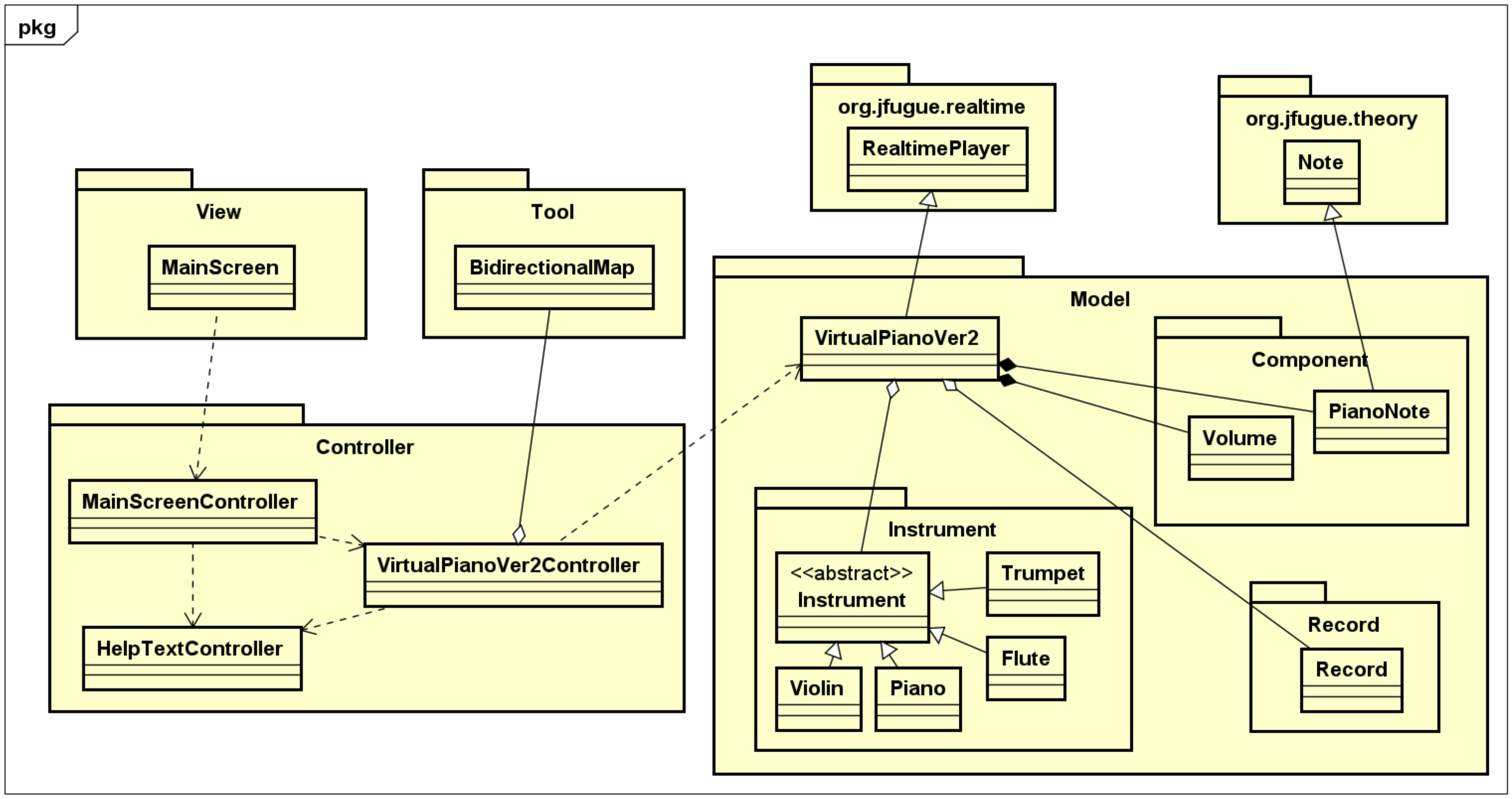

Figure 2: General class diagram

Packages:

- Package "Model": store a complete piano "VirtualPianoVer2" class, inherits from "RealtimePlayer" class of JFugue, with many classes as its components of piano, an abstract class "Instrument" to store common operations, attributes of 4 types of instruments, a "Record" class to store note, a "Volume" class to adjust magnitude of sound, a "PianoNote" class which inherits from Note of JFugue library, mapping note to octave ID in JFugue.
- Package "Controllers": store all the screen controllers.
- Package "View": store the "MainScreen" class of the application.
- Package "Tool": store key character and corresponding piano note, get each much more efficient.

3.2 PACKAGE DETAIL

3.2.1 VIEW PACKAGE

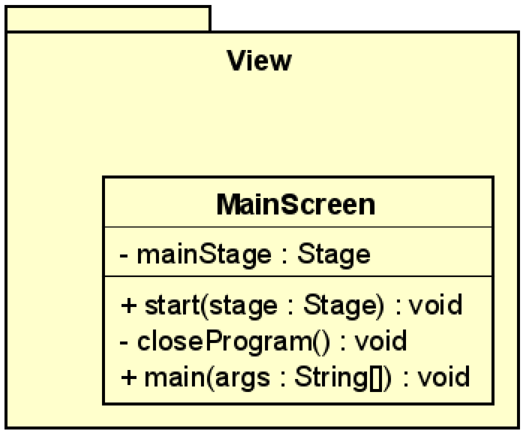

Figure 3: View package

This package is used for managing the main screen of the application.

For asking the user before exit we override the method setOnCloseRequest of the stage used for the main screen.

3.2.2 CONTROLLER & TOOL PACKAGE

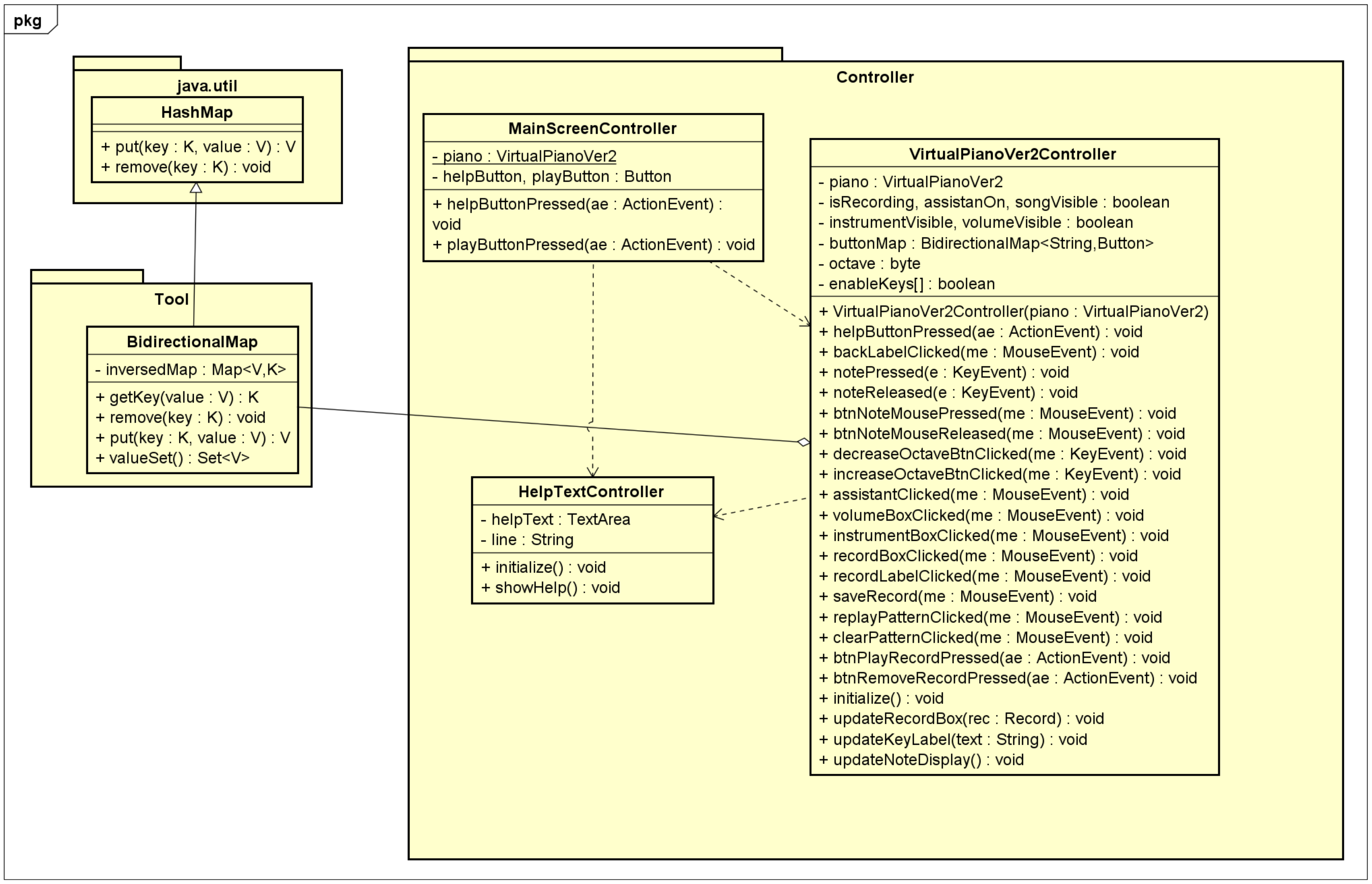
pkg
java.util
HashMap
+ put(key : K, value : V) : V
+ remove(key : K) : void
Tool
BidirectionalMap
- inversedMap : Map<V,K>
+ getKey(value : V) : K
+ remove(key : K) : void
+ put(key : K, value : V) : V
+ valueSet() : Set<V>
Controller
MainScreenController
- piano : VirtualPianoVer2
- helpButton, playButton : Button
+ helpButtonPressed(ae : ActionEvent) : void
+ playButtonPressed(ae : ActionEvent) : void
HelpTextController
- helpText : TextArea
- line : String
+ initialize() : void
+ showHelp() : void
VirtualPianoVer2Controller
- piano : VirtualPianoVer2
- isRecording, assistanOn, songVisible : boolean
- instrumentVisible, volumeVisible : boolean
- buttonMap : BidirectionalMap<String,Button>
- octave : byte
- enableKeys[] : boolean
+ VirtualPianoVer2Controller(piano : VirtualPianoVer2)
+ helpButtonPressed(ae : ActionEvent) : void
+ backLabelClicked(me : MouseEvent) : void
+ notePressed(e : KeyEvent) : void
+ noteReleased(e : KeyEvent) : void
+ btnNoteMousePressed(me : MouseEvent) : void
+ btnNoteMouseReleased(me : MouseEvent) : void
+ decreaseOctaveBtnClicked(me : KeyEvent) : void
+ increaseOctaveBtnClicked(me : KeyEvent) : void
+ assistantClicked(me : MouseEvent) : void
+ volumeBoxClicked(me : MouseEvent) : void
+ instrumentBoxClicked(me : MouseEvent) : void
+ recordBoxClicked(me : MouseEvent) : void
+ recordLabelClicked(me : MouseEvent) : void
+ saveRecord(me : MouseEvent) : void
+ replayPatternClicked(me : MouseEvent) : void
+ clearPatternClicked(me : MouseEvent) : void
+ btnPlayRecordPressed(ae : ActionEvent) : void
+ btnRemoveRecordPressed(ae : ActionEvent) : void
+ initialize() : void
+ updateRecordBox(rec : Record) : void
+ updateKeyLabel(text : String) : void
+ updateNoteDisplay() : void

Figure 4: Controller & Tool package

Tool Package:

- BirirectionalMap: This class extends from HashMap, allows using both key and value efficiently.

Controller Package:

- MainScreenController: This controller is used for managing the main screen of the application, where the user can click play to start piano screen, view help menu, or quit the app.

  + To ask before user quit the app, we have to get the current stage, use the method setOnCloseRequest(), and then set method "consume" to remain current stage if user don't choose "Finish", then we show an alert dialog box with FINISH_CANCEL options.
- HelpTextController: Creating another stage to show the application's user manual. User must turn off this stage before continuing to interact with main screen. There is a method for creating a new instance of this class whenever needed.
- VirtualPianoVer2Controller: This controller is created for managing all the JavaFX components and their corresponding action that we created in the fxml file.

- Attributes:
  - VirtualPianoVer2 piano: Create an instance of class VirtualPianoVer2, the action event from piano note will be used by this instance to make music.
  - boolean songVisible, instrumentVisible, volumeVisible: initialize visible state of container.
  - boolean isRecording, assistantOn: start or cancel report | add key character correspond to note right next to it.
  - BidirectionalMap<String, Button> buttonMap: Use to map from key character to button for setting style, fast implement others method.
  - Byte octave: initialize octave.
  - Boolean[] enableKeys: when pressing a note, the id corresponds to this note in enableKeys turns true, and when released, it turns back to false, allows user to play many notes at a time.
- Methods:
  - UpdateRecordBox(Record: rec): a separated method uses to check if the length of the song is enough to consider as a song, if not, it can't be saved.
  - UpdateNoteDisplay(): called from "assistantClicked" method, it set key character correspond to note right next to it.
  - saveRecord(MouseEvent: me): create a record and add to the song list of piano.
  - replayPatternClicked(): get string from keyLabel passing through piano, play every note.

3.2.3 MODEL PACKAGE

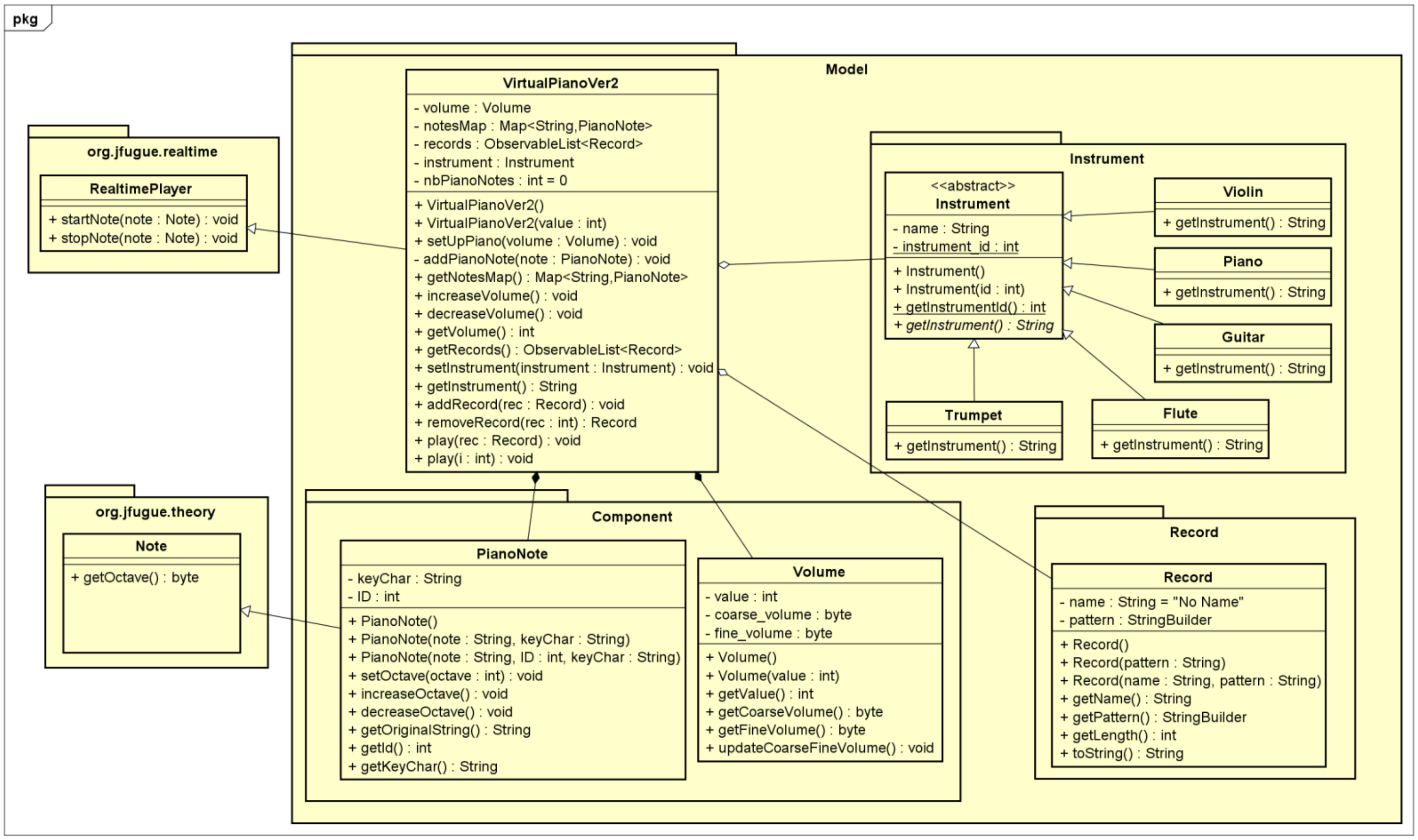
pkg
Model
VirtualPianoVer2
- volume : Volume
- notesMap : Map<String,PianoNote>
- records : ObservableList<Record>
- instrument : Instrument
- nbPianoNotes : int = 0
+ VirtualPianoVer2()
+ VirtualPianoVer2(value : int)
+ setUpPiano(volume : Volume) : void
- addPianoNote(note : PianoNote) : void
+ getNotesMap() : Map<String,PianoNote>
+ increaseVolume() : void
+ decreaseVolume() : void
+ getVolume() : int
+ getRecords() : ObservableList<Record>
+ setInstrument(instrument : Instrument) : void
+ getInstrument() : String
+ addRecord(rec : Record) : void
+ removeRecord(rec : int) : Record
+ play(rec : Record) : void
+ play(i : int) : void
org.jfugue.realtime
RealtimePlayer
+ startNote(note : Note) : void
+ stopNote(note : Note) : void
Instrument
<<abstract>>
Instrument
- name : String
- instrument_id : int
+ Instrument()
+ Instrument(id : int)
+ getInstrumentId() : int
+ getInstrument() : String
Violin
+ getInstrument() : String
Piano
+ getInstrument() : String
Guitar
+ getInstrument() : String
Trumpet
+ getInstrument() : String
Flute
+ getInstrument() : String
org.jfugue.theory
Note
+ getOctave() : byte
Component
PianoNote
- keyChar : String
- ID : int
+ PianoNote()
+ PianoNote(note : String, keyChar : String)
+ PianoNote(note : String, ID : int, keyChar : String)
+ setOctave(octave : int) : void
+ increaseOctave() : void
+ decreaseOctave() : void
+ getOriginalString() : String
+ getId() : int
+ getKeyChar() : String
Volume
- value : int
- coarse_volume : byte
- fine_volume : byte
+ Volume()
+ Volume(value : int)
+ getValue() : int
+ getCoarseVolume() : byte
+ getFineVolume() : byte
+ updateCoarseFineVolume() : void
Record
Record
- name : String = "No Name"
- pattern : StringBuilder
+ Record()
+ Record(pattern : String)
+ Record(name : String, pattern : String)
+ getName() : String
+ getPattern() : StringBuilder
+ getLength() : int
+ toString() : String

Figure 5: Model package

Main Idea: Instead of putting every piece in one class, we decide to divide the piano into smaller part, each correspond to one property of a sound (including pitch, amplitude, timbre). Then we will combine all of these in a main class, and implements the "play" method on this main class.

Instrument Package: Each instrument inherits from abstract class Instrument, they must implement the abstract method getInstrument, each of them have a specific id in JFugue

Component Package:

- ❖ Volume: this class will translate value which human can understand to value JFugue can use.
- ❖ PianoNote: this class inherits from "Note" class of JFugue, get the current octave and can set new octave.

Record Package: Record class store default song and can be added new song in form of StringBuilder.

VirtualPianoVer2: From all the class in package above, this class create a complete piano. It inherits “RealtimePlayer” class from JFugue. In depth:

- ❖ SetUpPiano: Map every note to a specific key, set volume and id for each note.
- ❖ play: Take record from records and use for the play method from RealtimePlayer.