Hanback-Launchpad

종합설계

32122630

이서호

32112201

송준형

Index

1. Introduction
   1. Requirements
   2. Schedule
2. Design
   1. Ver\_1
   2. Ver\_2
3. Programming
   1. Kernel
   2. Core
      1. Controller
   3. JNI
      1. JNI interface
      2. Key Pad
      3. Dip Switch
      4. Text LCD
   4. Main Activity
4. JNI shared library building
5. Project result
6. Future
7. Conclusion

**1. Introduction**

This application is to build an working launchpad(DJ Pad) using Hanback electroinc’s HBE-SM5-S4210.

A. Application requirements

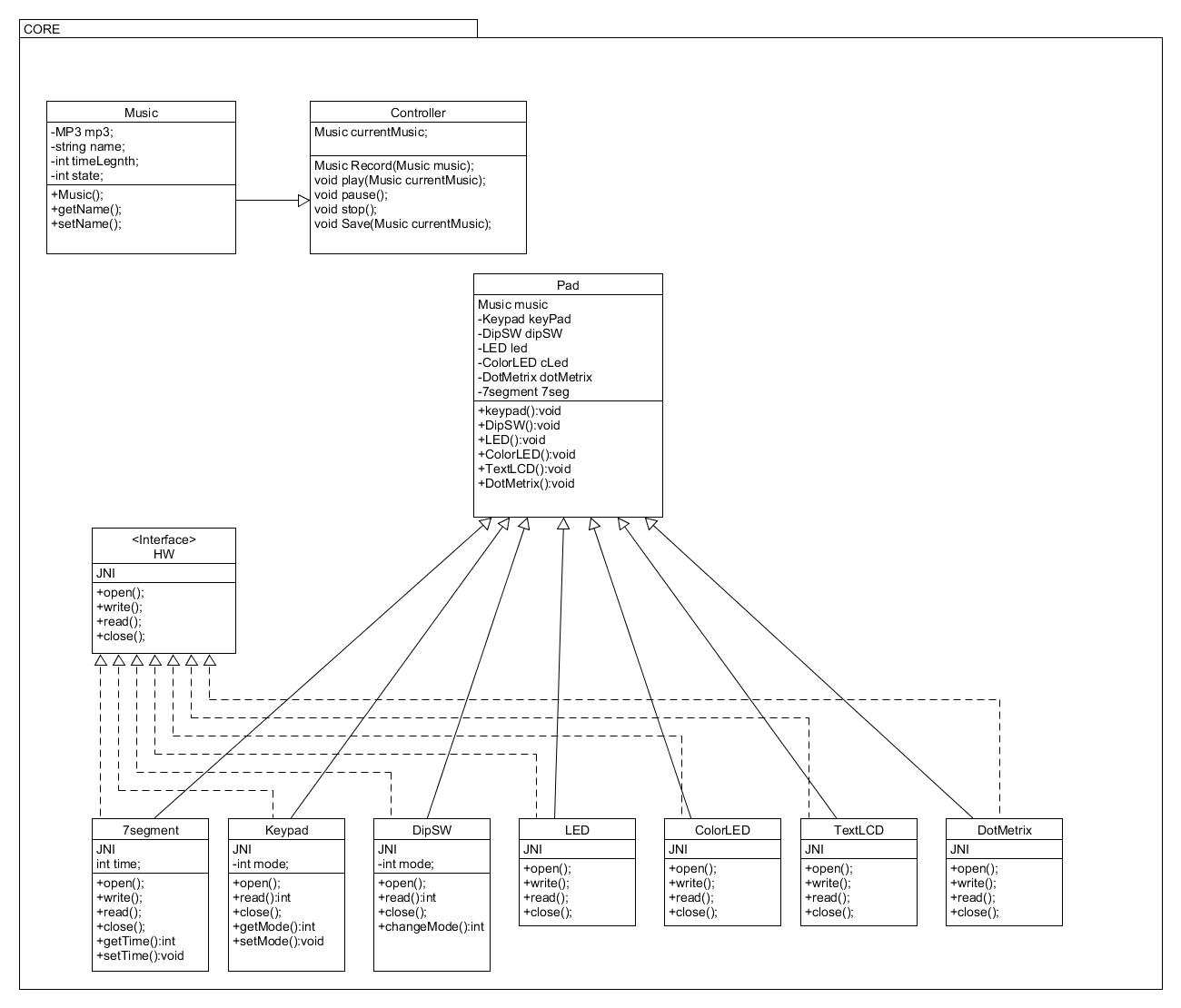
1. Use keypad to make sound. Each keypad button will make different sound corresponding to current sound mode.
2. Use Dip-switch to change sound mode.
3. Use Text LED to dispaly current sound mode.
4. Allow user to record, play, stop, and pause the sound.

B. Schedule

1. UML diagram disign …………………………………………...…………….Due on May 26
2. Prototype application ……………………………………………..………...Due on June 2
3. Finalize application …………………………………………………………...Due on June 11
4. Final report ………………………………………………………………………..Due on June 14

2. **Design**

1. **Ver\_1**

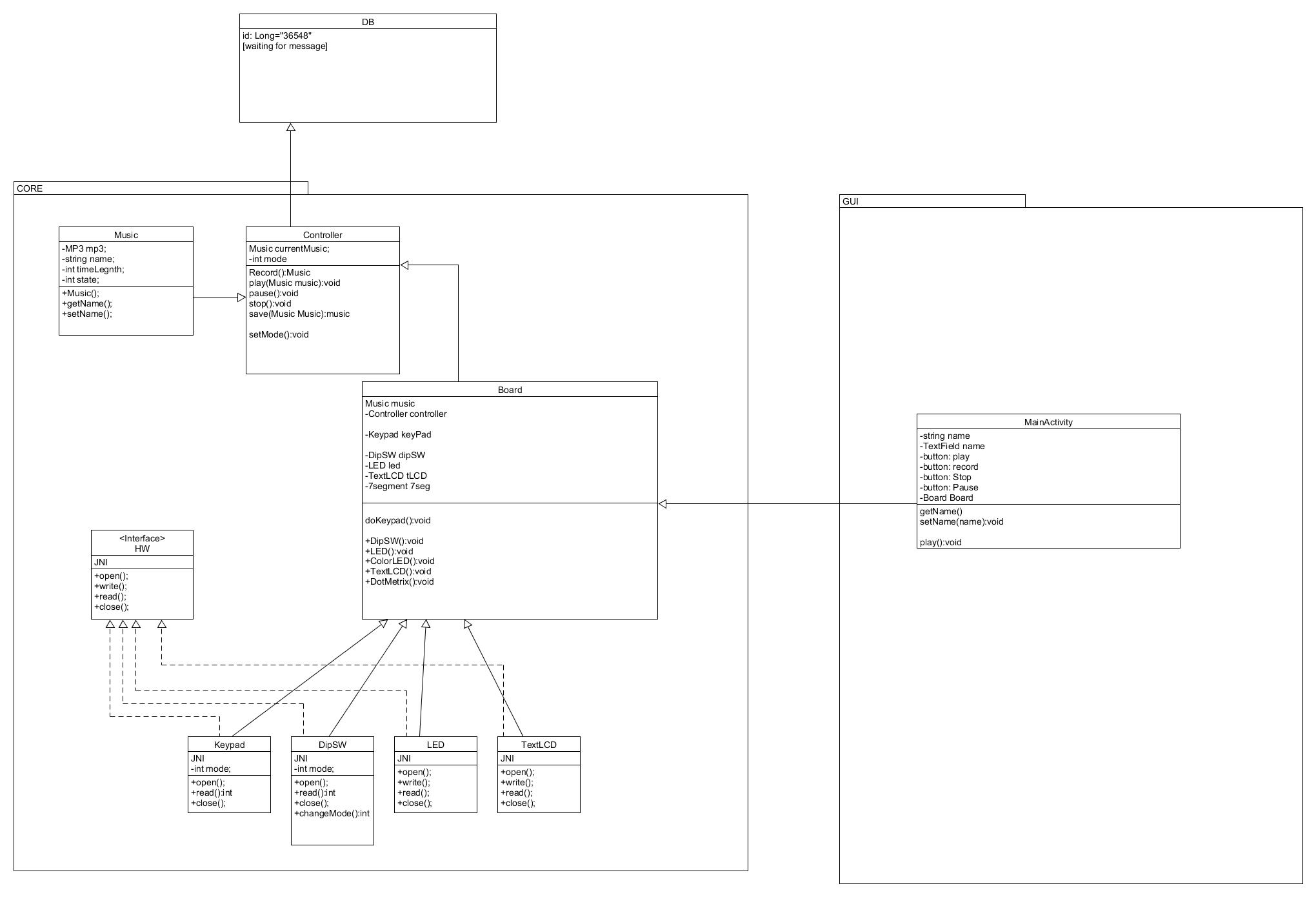


Our first class diagram. In this diagram it’s containing basic requirements and ideas. The basic idea is to resemble the Hanback android pad itself. So as it is shown in the diagram it has Pad class holding individual hardware classes. The hardware classes are implementing HW interface and it should be using JNI since the hardware are readable using C code.

The Pad class also holds Controller class to play sounds and manage modes and save the recorded music. The controller gets values from hardware classes and execute functions according to the values.

Also, at first, we made Music class to hold mp3 source file, name and length so we can define music, record and save.

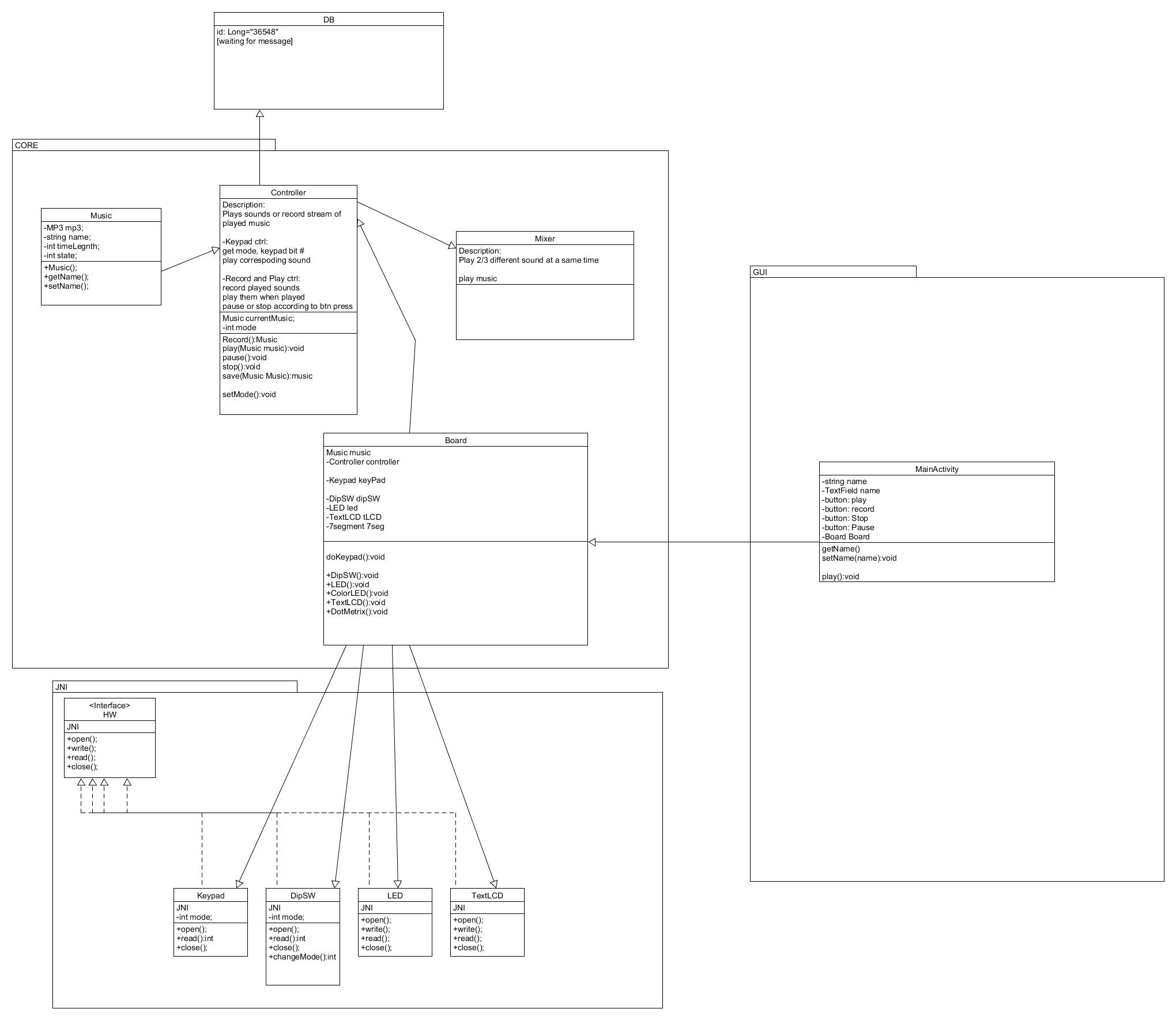
1. **Ver\_2**



In the second design we have included MainActivity which is class for GUI in our application. MainActivity holds Pad class so the Android can work with the classes in the core package. This resembles MVC (Model-View-Controller) pattern.

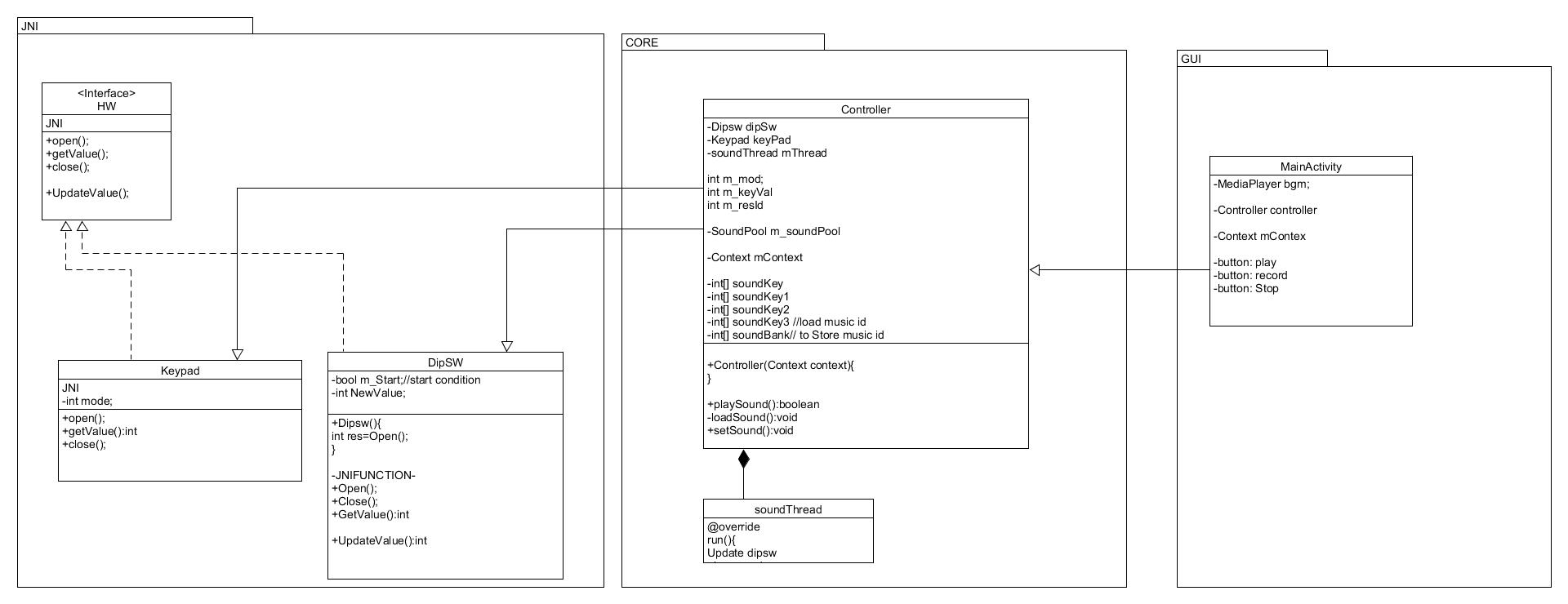
Also, this design has data base module connected to the controller, since the controller is the class that plays and save sounds from the database. The controller will get sounds from the database and play, and it will save new music to the database.

1. **Ver\_3**



The Third version of our class diagram is we have separated the JNI files into separate package.

1. **Ver\_4**

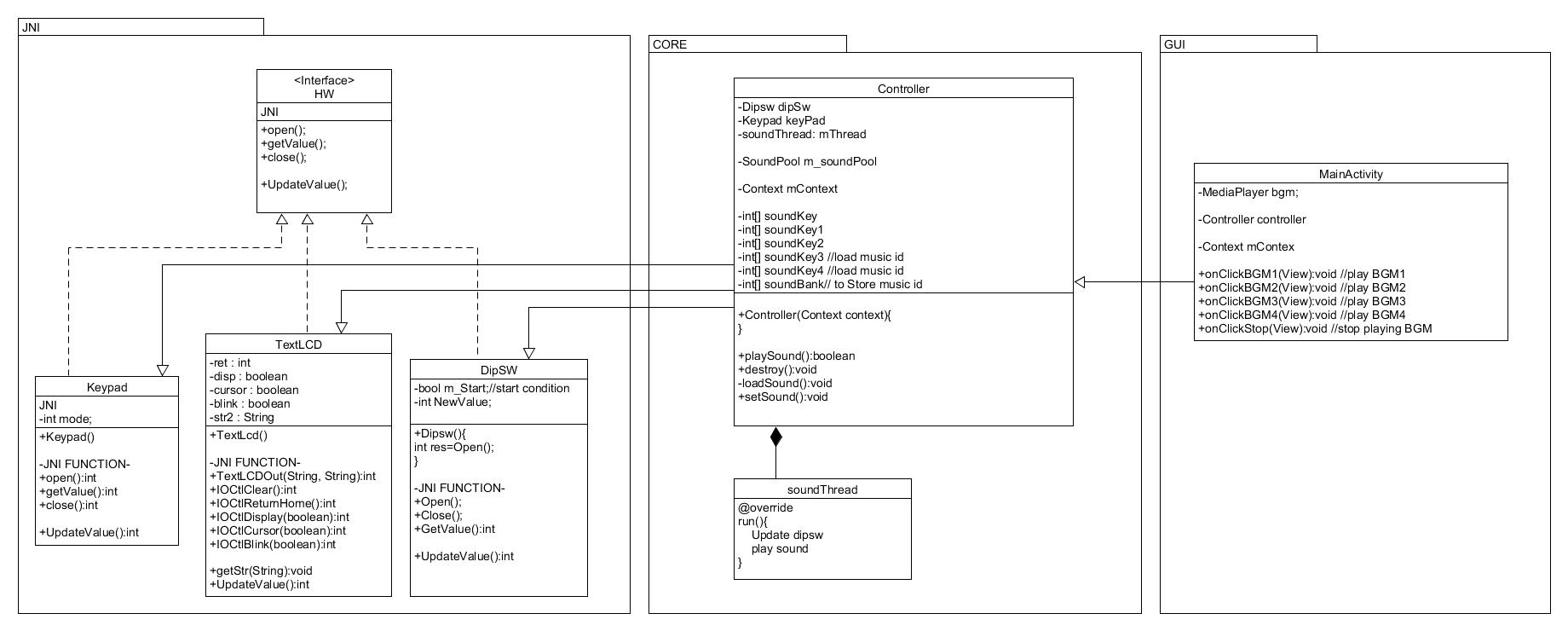


As we studied Android and lack of time we had to minimize and choose what we can actually make. So we have finalize to make keypad and dipswitch as our main function to the application. Also, we have discarded Database since it is hard to implement and takes longer time to fetch sound sources due to overhead between classes and database. We have made more clear design of MVC design by getting rid of the pad class. We have realize that pad is useless since it is the Controller doing all the actual job. The Controller has inner class, SoundThread, to continuously check the keypad input.

So our diagram works as follow.

1. MainActivity gets started by Android and onCreate function is called.
2. In onCreate the Controller will be created and it will also create keypad and dipswitch classes.
3. After the controller is created the thread in controller will continuously check the keypad.
4. if keypad is pressed it will call setSound which will check the current mode by checking dipswitch and play the corresponding sound.

1. **Ver\_final**



The final class diagram we have added Text LCD class since it is very easy to implement and will enrich our application. Also we have fully discripted functions we need.

**3. Programming**

1. **Kernel**

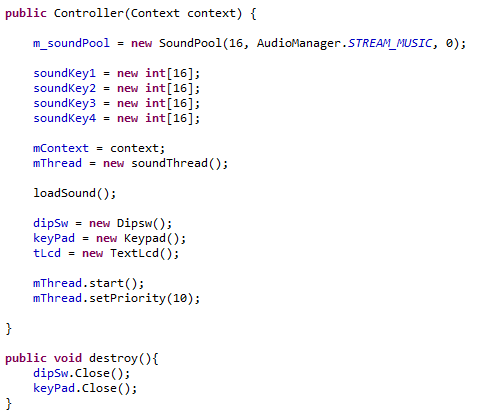
Our first problem was that keypad wasn’t returning the right values. The reason was that in the device driver the values were declared according to Android properties, which was not what we were expecting. So we have changed the kernel source code.

|  |  |  |
| --- | --- | --- |
|  | **→** |  |

Kernel should be built with cross compiler since our building environment is running on Intel CPU and the android is running on arm cpu. After the build the zimg should be flashed with fastboot.

1. **Core**
2. **Controller**

As the class name shows, this java file controls the Sound and Music play. It works by listening to the Keypad or Dipsw to play different sound or different play mode. Andshow the current status by displaying it on the Text LCD.

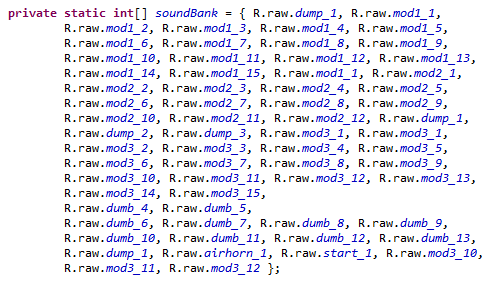


* Constructor & Destructor:

Constructor sets SoundPool class to allow multiple sound plays, sets sounds to the 4x4 keypad through loadSound() method. And instantiate dipsw, keypad, tlcd, thread and starts thread that sets mode or keypad digit. Destructor closes the keypad and dipsw instances.

* soundBank:

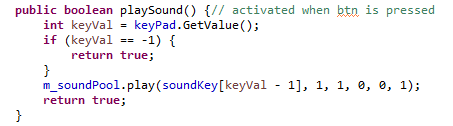
integer arrary that stores the IDs of the sound files.



* playSound:

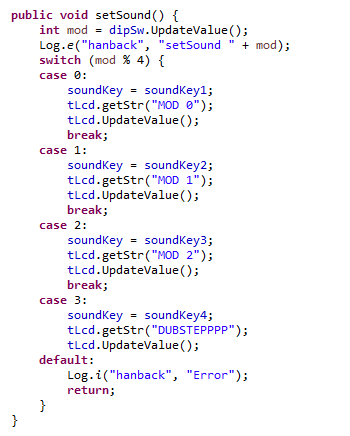
This method plays the corresponding sound when the keypad’s key is pressed. It’s operated in the thread that listens to the keypad’s values and when the button is pressed through keyPad object, it gets the value from GetValue method. then soundpool plays the corresponding sound through the speaker.

in exception handlng, sometimes it had problem reading the keypad signal so when the error is noticed, we just ended the operation until the next feasible signal was noticed.



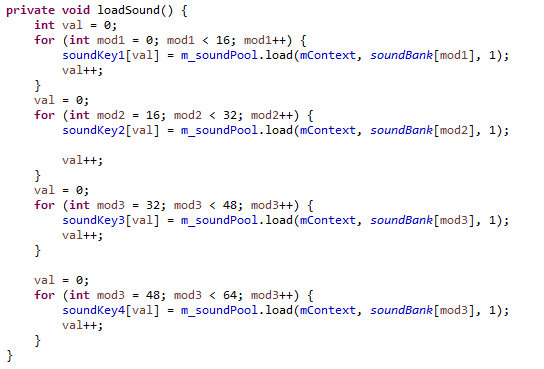
* setSound:

This method sets the mode of the sound played depending on the dip switch’s condition. When the keypad is pressed it first callse setSound to set the sound mode. It gets dip switch value and sets the different soundKey\_ array to soundKey array. For now, we only have only 4 modes. So we placed modular 4 to assign mode. While setting the soundKey, we update the text lcd to show the current sound mode.



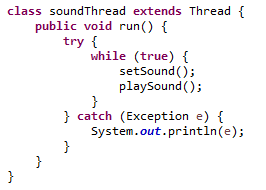
* loadSound:

This method loads the sound to the soundKey\_ array. It is to previously set 4 key arrays to different sounds according to the different modes. We implemented it by looping through the soundBank and assign the IDs of the individual sounds. So this method is called at the starting of the operation.



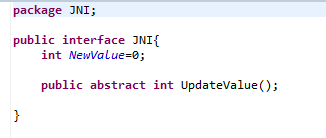
* soundThread:

This method runs the subordinate thread to call setSound method and playSound method. It is to catch the changes of the dip switch condition and keypad signals. So, it could set or play the sound according to the changes made.



1. **JNI**
2. **JNI interface**

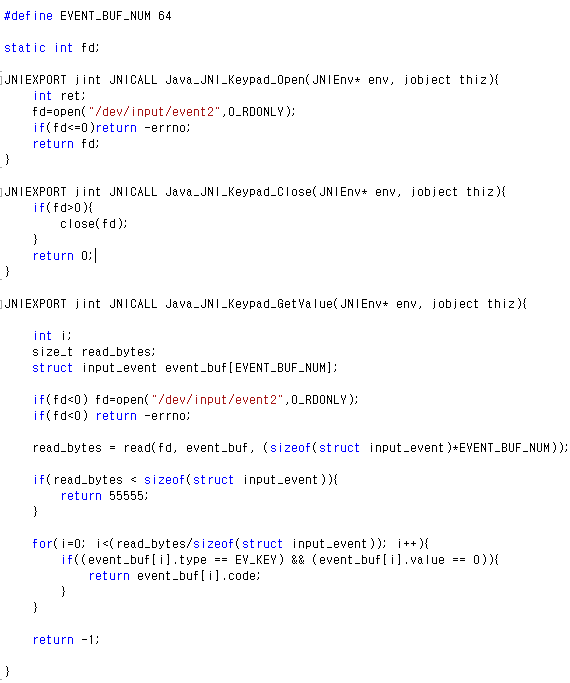
This is an interface to define what Java files that uses JNIs. It holds new value to save values from the hardware and UpdateValue function to update the new value.

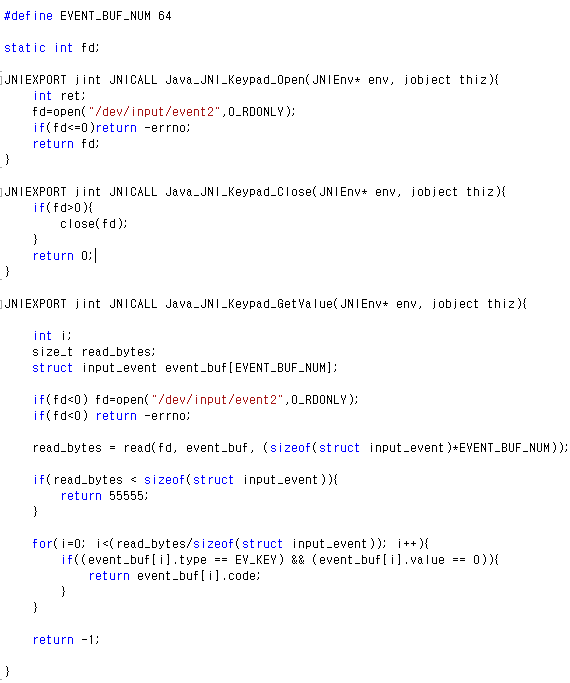


1. **Key Pad**

* keypad.c:

This is C code that handles actual system call. This is written with c code since it needs access to the driver. This file is connected to KeyPad.java by JNI(Java Native Interface).



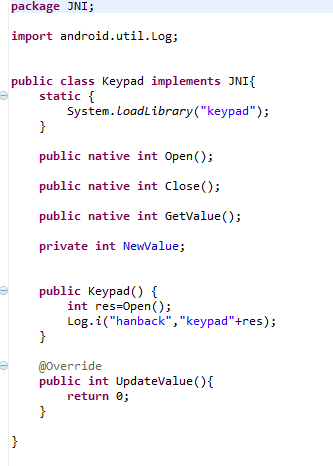


JNIEXPORT and JNICALL is used to link the function to KeyPad.java, if not it shows that the functions can’t be found. Also, the function name should be declared right using package name, java file name, and function name so the JNI can link functions. In this case, JNI\_Keypad\_Getvalue, JNI is package name, Keypad is java file name, and Getvalue is function name.

1. Open function is to open device file “/dev/input/event2”, which is used by key pad driver.
2. Close function closes the file.
3. Getvalue function reads data from the “/dev/input/event2” and returns right value among the read values.

* KeyPad.java:

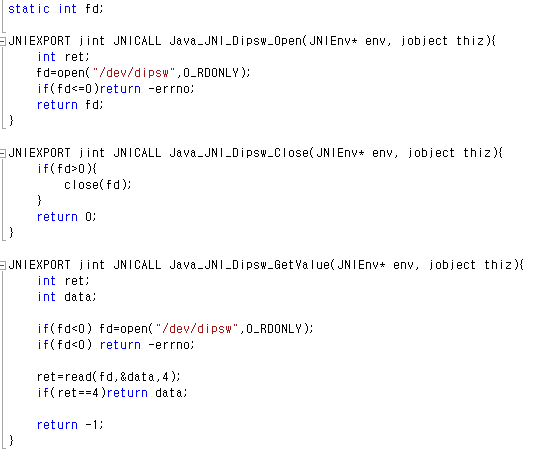
This is class file to define Key pad function and define JNI native functions. This class is connected to keypad.c by using JNI. When this class is created file “dev/input/event2” is opened. But the UpdateValue is not used, since the Controller will get values directly using GetValue function.



1. **Dip Switch**

* dipsw.c:

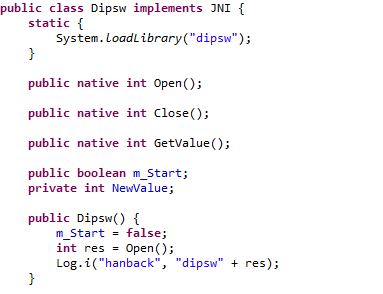
This file is to declare c source to read dip switch value. This source first open the device file, “/dev/dipsw” and reads file to get the dipswitch value.



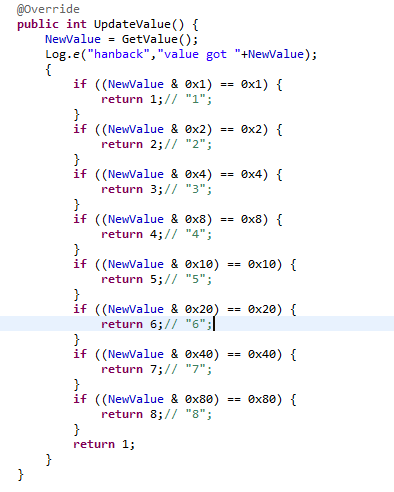
1. Open function opens “/dev/dipsw” file.
2. Close function closes the device file.
3. GetValue function reads the device file and returns read data.

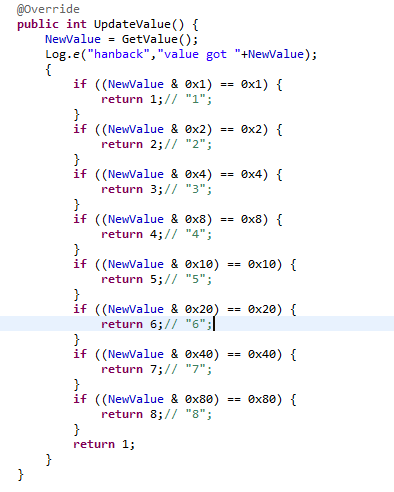
* Dipsw.java:

This is class file to define Dip switch functions and define JNI native functions. This class is connected to dipsw.c using JNI. When this class is created, file “dev/dipsw” is opened. Then it reads the file to return the data. Also, the UpdateValue handles read data and returns corresponding number.



In this class the JNI native function methods are first declared. When this class is created it opens the “dev/dipsw”.



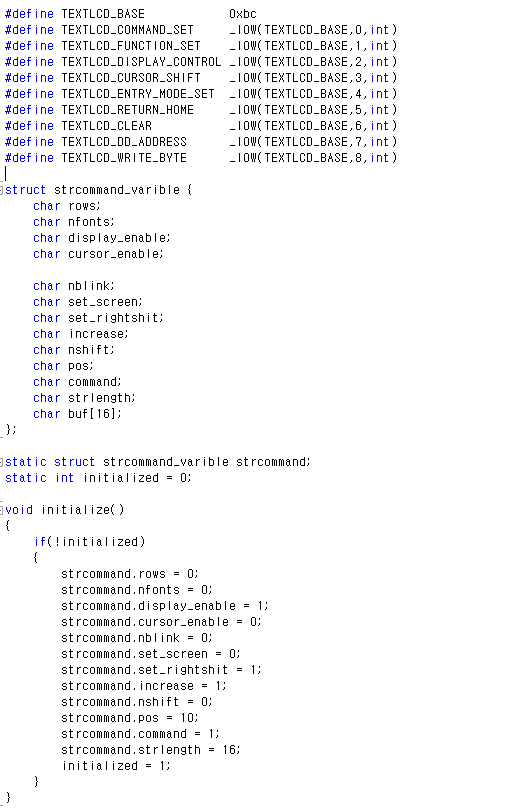


UpdateValue function is to read value from the device file and process it to return the right number. So it first gets value and use if states to return the right value.

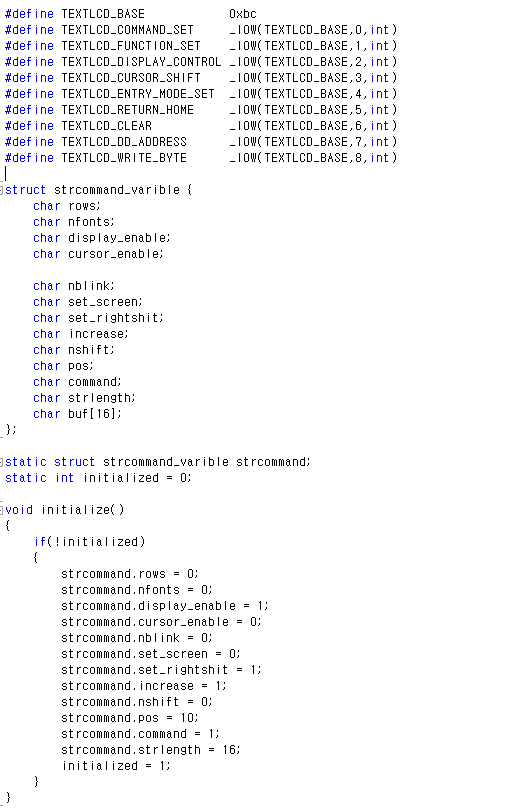
1. **Text LCD**

* textlcd.c:

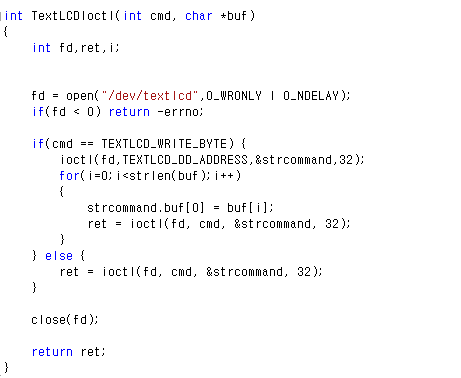
This file is to declare c source to manipulate text lcd. Text Lcd c code is bit different from Dip switch or key pad since it uses IOCTL to directly manipulate the hardware.



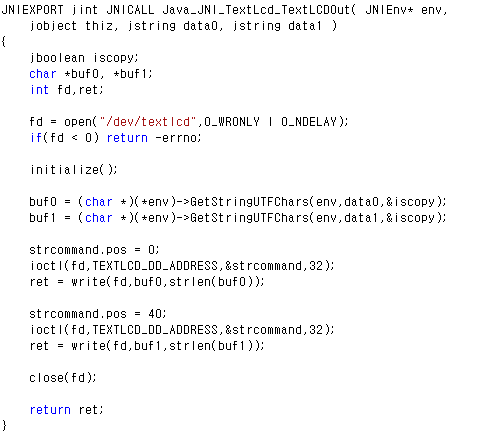
Text Lcd is controlled with iocltl and it needs actual commands to be declared, these commands should be identical to the commands declared in the device driver. The structure to manipulate Text LCD register is also declared.



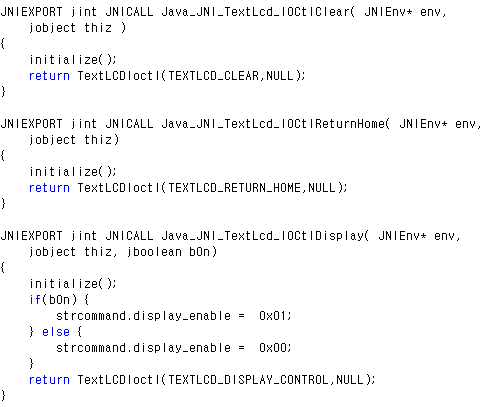
Than we made function to initialize each register.



The TextLCDIoctl function is to send different commands to the device driver using ioctl function.



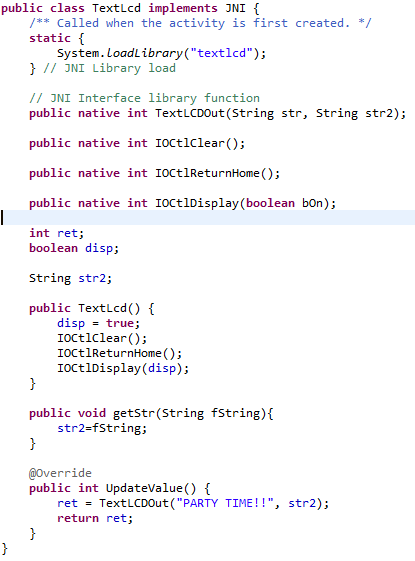
This is function that prints string to the Text LCD. Our board’s Text LCD has two lines, so it receives two strings as separate arguments and send them to right address to print out two lines.



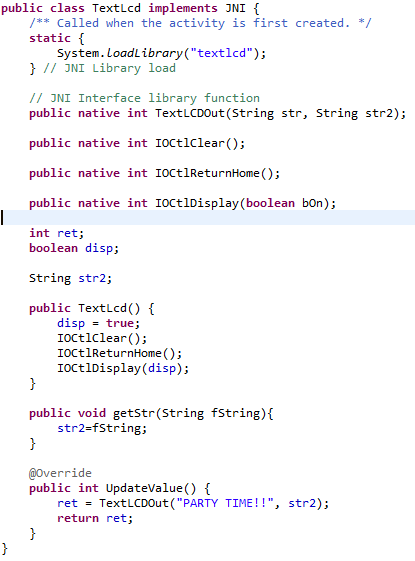
IOCtClear is to clear the Text Lcd. IOCtlReturnHome function moves the cursor to the initial position. Finally, IOCtlDisplay turns Text LCD on and off.

* TextLcd.java:

This is class file to define Text LCD functions and define JNI native functions. This class is connected to textlcd.c using JNI.



First the shared library created by textlcd.c is loaded to the class and the JNI native function methods are declared. String to use in this class is also declared.



When the class is created the text LCD is initialized by disp becomes true and LCD is cleared, cursor is set to initial position and turn the LCD on.

getStr function sets string in the class then Updatevalue writes it on the text LCD.

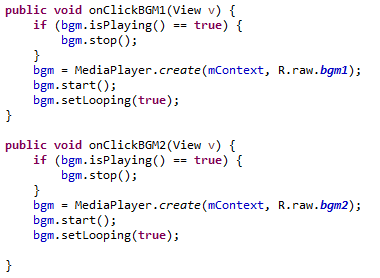
1. **Main Activity**

As the application starts, it sets the initial conditions. Its main function is to instantiate the Controller object. And to create buttons that would play 4 different background music and 1 stop button.

* onClickBGM\_:

This method is called when the corresponding button is touched. When the method is called, it checks whether the background music(so called BGM) is playing, if BGM is not playing, then it sets wanted BGM and plays it. We also made the BGM to loop so even though the music is ended it plays it again. In the other hand, if themusic is already playing, then it stops the music. So many BGMs would not overlap.

* onClickStop:

This method is to stop the background music

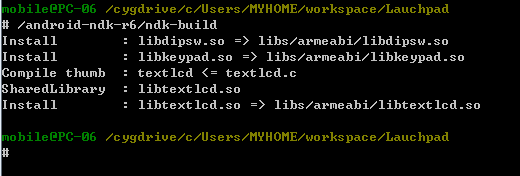
**4. JNI Shared library building**

To use the hardware attatched to the board, we used JNI(Java Native Interface). To use JNI we had to build shared libraries that can be accessed by our android application. Here we used program called “Cygwin” to build shared library. However, we had little trouble using it since the execution file had wrong directory and awk version was wrong.

To use C files in Android we need C code, which was explained earlier, and Android.mk file.

|  |  |
| --- | --- |
|  |  |

Android.mk file sepecifies the modlue name and makes sure it is built into shared library.



After all the files are ready. We run Cygwin to build shared library. If build is successfully done, you will have libraries ready in the lib folder of the project.

**5. Project result**

|  |  |
| --- | --- |
|  |  |

The main screen follows the main.xml and MainActivity.java. As it’s shown on the right, when the button is pressed, it first reads dip switch value and sets sound. Then it plays the sound.

**6. Future**

For the future addition, we are thinking of adding record button and play button. Which will record the music that we are playing through our launch pad, and play the recorded music. Because of the shortage in time and knowledge on this functions, we couldn’t implement them on time with our final set.

**7. Conclusion**

Over all, the Hanback launch pad was very successful project. In the beginning we started off with good class diagram fallowing MVC pattern. It was hard to make clear UML diagram due to lack of knowledge in UML designing. However the final version seems to be very clear of how this application works.

Since we have made our class diagram it was easy to separate tasks. The tasks were divided according to its packages, core, JNI, and GUI. The core and GUI was handled by Junhyeong and JNI was handled by Suhho. Since we had our planning done it was easy going since then.   
However we have still faced some difficulties. For the Core we had to use Android functions such as Sound pool and Media player. Due to our shortage of knowledge it was hard to use them properly at first. For the JNI it was hard because of more general things. We had problems with the compiler settings and thread handling. But, we have managed such problems by searching internet and minimizing thread usage. Also, we had problem with the sounds having latency. This was also solved by minimizing thread and removing delay.

This application project was very interesting to work with since we had to use lots of different things. First, we have manipulated with kernel, then build JNI files using both c and java. Finally we also have used APIs in Android SDK, such as Sound pool and Media player. Through this project we were able to make an application that runs from very bottom of Andoroid to the highest level.