CS446 Deliverable 5 - Architecture and Design Document

Project Title: Pitch-Perfectly-Accurately Practice

Irvine Yao (b6yao)
Jialin Shan (j6shan)
Alex Lai (x7lai)
Johnny Gao (z53gao)

.

1 Architecture

1.A Overview

In order to fulfill our apps main functionality, the following general architecture logic of our application is the following. First, we display the questions according to the modes users are currently using. The user can further customize what questions that they want to have asked by using the filter page. Then user will try to sing the correct note that is asked of them. Our app will constantly listen to the user through listeners with the mic. Once we got the sound data we will send the data to our library to get the current frequency to check whether user is singing the correct note or not. During the sound detection, we will display a bouncing arrow, asking the user to sing higher or lower. Based on our how our algorithm processes the data sent back from the library, we will display the result accordingly and if the answer is correct, we will display the next question. Otherwise, user can switch mode freely between note practice, interval practice, and triad mode, and our question page will be updated and generate question accordingly.

1.B Model View Controller

In order to handle the sound data transmission between users, the result checking algorithm, and the library for sound frequency, we implemented the Model View Controller architecture as our main backend structure.

In our MVC architecture framework, Controller is the component where most of the applications algorithms and logic reside, and is able to manipulate model and update view. Model component have all the data and objects we need for the data transmission within the app. It is inert component where if no one calls it, it will not do anything, and provide data to controller. View is a component that can interact with user, having layouts, views for displaying information, Microphone for gathering sound data, and Noteplayer for playing sound to user.

The reason we choose MVC architecture style for our app is because our app is highly interactive. So in order for users to see real time respond right when the user produce voice, we need model can help us store history of what user sings, and having the controller as the central processing unit enables it to modify model and update view in real time. Model-view-controller is the best choice.

1.C MVC Application

Once our application has started, our Controller component will generate the question based on our default settings data from the Model component, then send them to the viewer component of our architecture. In our view component, we will set up layouts and text, and a runnable thread in our VoiceListener class by utilizing a method from the Tarsos library, the library we use to handle the sound signal data. The library creates a thread that will be used for constantly getting data from the microphone and generate the correct current frequency from the data. The user will see the question through the Viewer component and try to provide the answer to controller component, by singing to the microphone connecting

to our view component. Once the view component get the sound data, it will generate the frequency of the current sound and send this data to the Controller component in real-time for frequency processes. In our Controller component, we have multiple internal logics and algorithms that will decide whether the current note our user is singing is too high, too low or is in the target frequency range, and will update the viewer component in real-time accordingly.

Also, In order to let the user to navigate between different modes, we created a left drawer containing 3 buttons corresponding to the modes in the viewer. Each mode will be represented as a Fragment in the Controller. Once the user wants to switch mode, they will interact with our user interface through their touch screen in the View. The view will then notify the Controller component which mode is requested, from which Controller then will update viewer correspondingly.

A Microphone (view) is where user can input voice. Controller gets notified whenever current frequency stored in Microphone is changed. Controller then process the frequency and updates view the arrow suggestion (Note Practice Mode) or a real-time graph (Graph Mode).

Scenario: The user clicks on Mode button (e.g. Note Practice Mode) in navigation menu (view), the view notifies MainActivity (also controller). The MainActivity then change current Mode in Model. Since Controller (not MainActivity) subscribes to data changes in Model, Controller gets notified and then further interact with the Model. Finally, Controller updates view with texts from next Question.

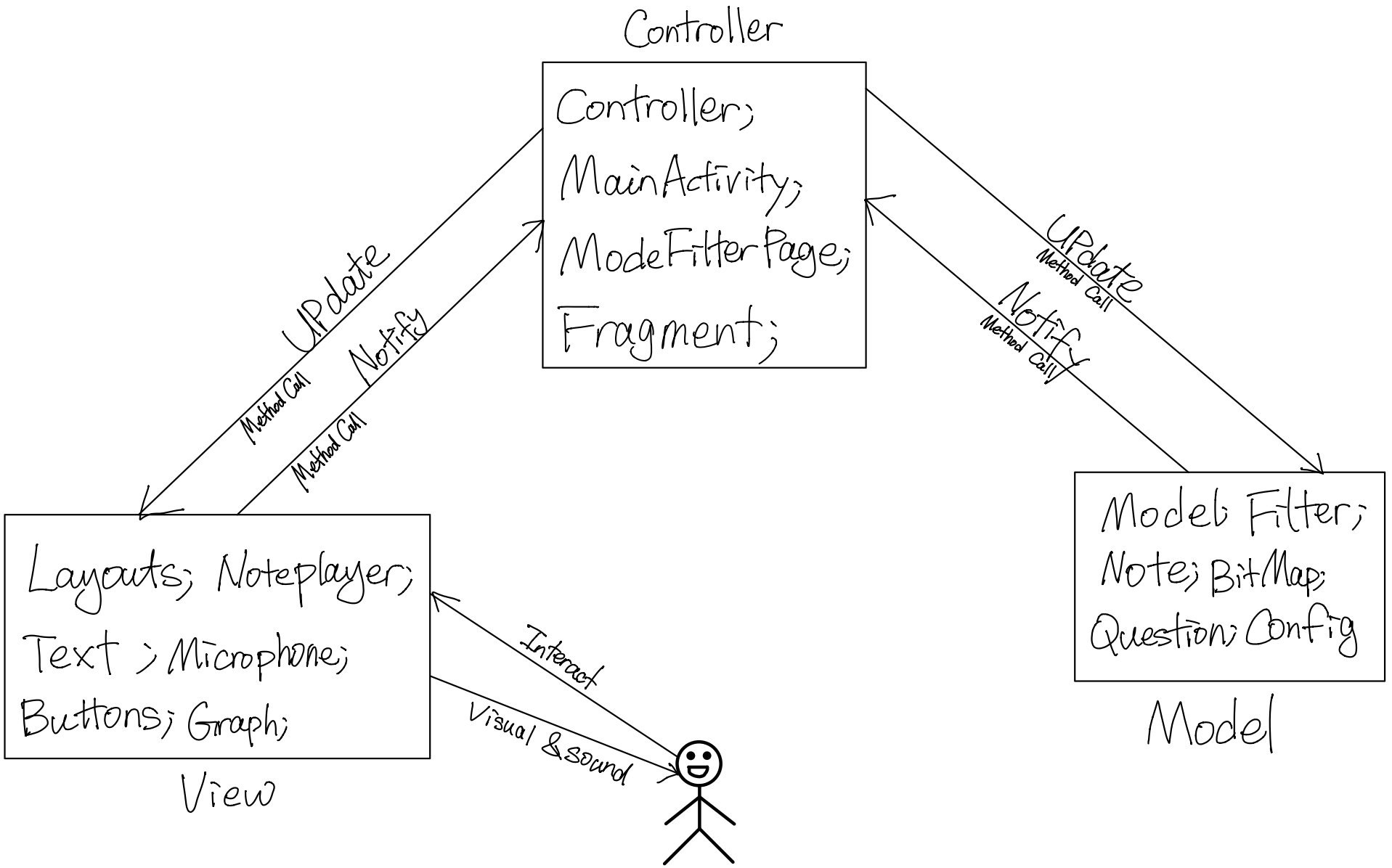
1.D Pipe and Filter

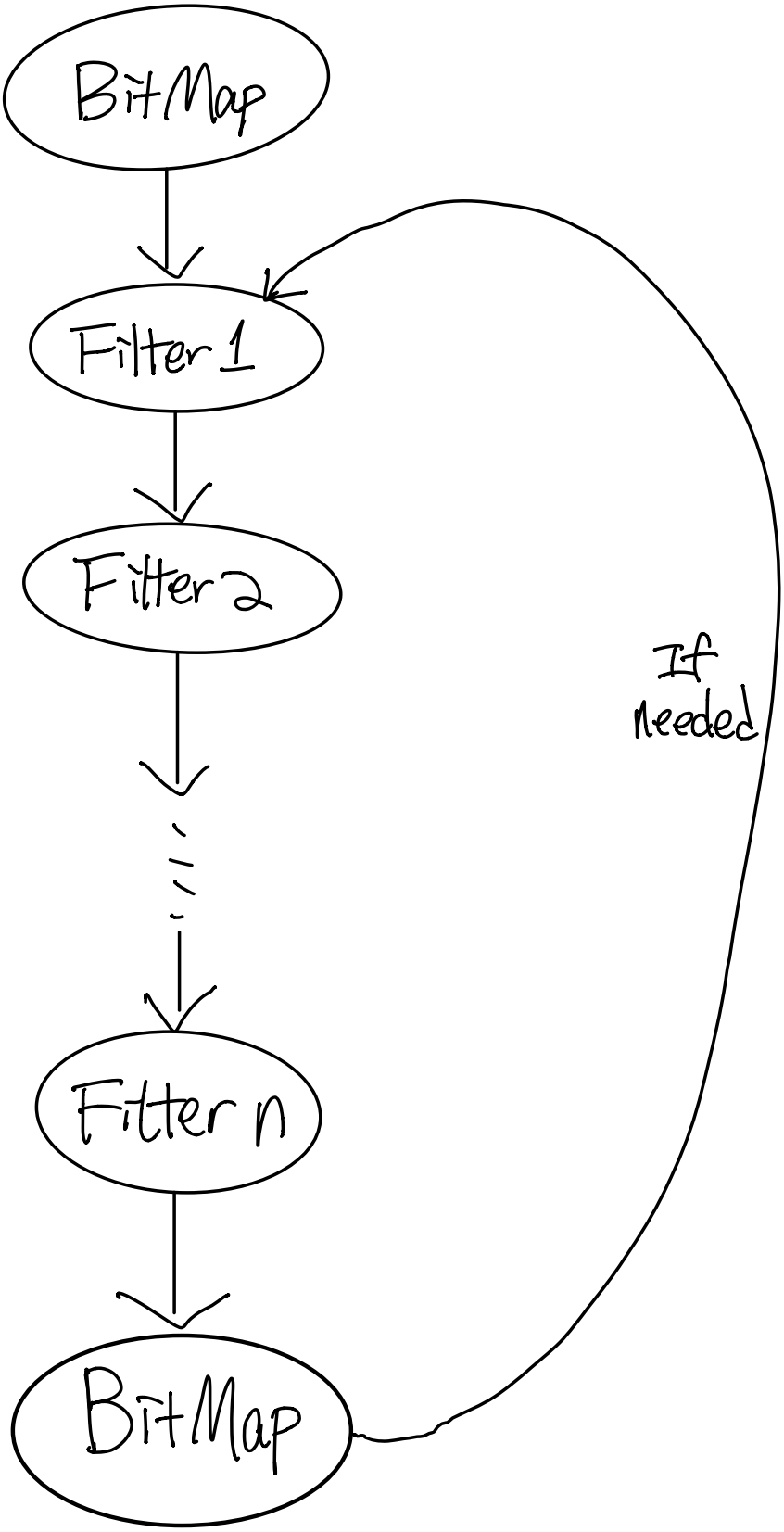
To implement filter page, we utilize pipe and filter (figure) as architecture style. At first, we have a bitmap with 73 1s, representing all 73 notes, which are all the musical notes available. Once user finishes selecting their preference on different spinner on the filter page, we pass our initial bitmap to a series of filters one by one to get the final result, like a pipe line. For example, if you are in note practice modes filter page, our backend will automatically have two filter, range and scale filter, ready. Then you will select the range of your preferred notes, scale and key signatures of the notes you want to practice. Every time you make a selection, the bitmap will go through those aforementioned filters to get the updated note pool, and display under the spinner. Finally, after user clicks the note that they dont want to sing, we will have the result bitmap with all the note that has passed the filter, and waiting to be chosen as question.

1.E Others

For our question generation, we use inheritance class structure and polymorphism to manage our question classes. We first have an abstract question class containing all of the reusable methods and fields for every modes question. Then we implemented three question classes as the child class for each practice modes question.

Our architecture design is also able to support most of the NFPs, such as scalability, maintainability and performance, aforementioned in the proposal documents. Thanks to the modularity of our overall architecture design, the scalability maintainability part of our NFP can be easily achieved. For example, If we would like to add more practice modes to our app later on, we could simply create corresponding fragments and their question class under their own higher abstract class, and that would be it. Also thanks to the advantage of MVC, we can maintain each component of MVC independently without disturbing the whole system. Last but not least, our library allows our app is able to give real-time feedback while user singing without regard to your devices, achieving the performance and usability.


Controller
Controller;
MainActivity;
ModeFilterPage;
Fragment;
Update
Method Call
Notify
Method Call
Update
Method Call
Notify
Method Call
Layouts; Noteplayer;
Text ; Microphone;
Buttons; Graph;
View
Interact
Visual & sound
Model; Filter;
Note; BitMap;
Question; Config
Model


BitMap
Filter 1
Filter 2
Filter n
BitMap
If
Needed

2 Design

2.A MVC Related Classes

2.A.1 Enum

- Mode : including NotePracticeMode, IntervalPracticeMode, NoteGraphMode, IntervalPracticeMode
- NotesScale: Major, MelodicMinor, HarmonicMinor, NaturalMinor

2.A.2 View

- Microphone stores a frequency as a float buffer. After construction, it starts a UI thread to get raw data from Android driver at certain sample rate, and (with the help of Tarsos library) to transform into a frequency. Then the frequency is stored in the buffer.
- NotePlayer

2.A.3 Model

- Model stores current Config, current Question, current General Fragment, current Mode. And it can notify observers who observes the property change of these objects.
- GeneralFragment extends Fragment, which is the parent of four children (currently), i.e. NoteFragment, IntervalFragment, TriadFragment, NoteGraphFragment.
- Bitmap is essentially a boolean array, which is the parent of two children (current), i.e. NoteBitmap, IntervalBitmap. It is mainly stored as internal data in Filter.
- Note is a primitive data in our app. It internally is an integer index (from 0 to 72, i.e. A1 to A7). It can be represented as a String (e.g. A4), an index (36) and a frequency (440 Hz). And these three types (representations) can be converted among each other.
- Interval is a primitive data in our app. It internally is an integer index (from 0 to 24, i.e. from negative perfect 8th to positive perfect 8th). It can be represented as a String (e.g. + P5), and an index (e.g. 19).
- Triad constitutes of three Note.
- Filter is an abstract class of two children, i.e. NotesFilter, IntervalsFilter. It stores a bitmap. It can be applied by other bitmap which use bitmapAnd (basically element-wise and) to get the result bitmap.
- NotesFilter is also an abstract class of two children, i.e. NotesRangeFilter, NotesScaleFilter.
- NotesRangeFilter can be created by specifying a fromNote and a toNote
- NotesScaleFilter can b e created by specifying a keySigNote and a NotesScale
- IntervalsFilter can be created given a bitmap
- FilterHandler stores an array of filters and initial bitmap. When it is called by a controller to apply the filters with initial bitmap. The bitmap gets filtered by the filters. The result bitmap can be get using getResultBitmap.

2.A.4 Controller

- Controller does four things. It listens to views. It listens to model changes. It update views. It modifies Model. The main logic of how a frequency gets processed and updates views lies in Controller.

- MainActivity owns a Model, a Controller, a notePlayer, a Microphone. It implements navigation menu listener. It handles intents passed from NoteFilterPageActivity.
- NoteModeFilterPageActivity is a filter page which lets the user to select a set of Note as a pool to generate NoteQuestion. It listens to four spinners (fromSpinner, toSpinner, scaleSpinner, keySigSpinner) and generate an array of Note. It (as a controller) then update view with an array of toggle buttons which has text on it (e.g. A4). It also listens to the backButton on the view. When user clicks on the backButton, it sends the array of Note back to MainActivity as integer array (as an intent) from the note array.

2.B Classes Communication And Scenario

Scenario 1
- In MainActivitys construction and onCreate, it initializes Microphone, notePlayer, Controller, Model.
- In Models construction, a Question, Config, GeneralFragment, Mode is created.
- In Controllers construction, it tells FragmentManager to replace a tlContent layout to current Fragment which is stored in Model. Controller also starts to observe Microphone.

Scenario 2
- Microphones UI thread gets a frequency.
- Controller gets notified by Microphone.
- Controller then evaluating OffTrackLevel(s) using currentFrequency and expectedFrequency(ies).
- Controller uses OffTrackLevels get_ArrowSuggestion() to get arrow texts.
- Controller calls GeneralFragments updateQuestionTexts() to update arrowTextView which can be accessed by currentFragment

Scenario 3
- MainActivity listens Navigation menus onItemSelected events.
- User selects a menu item (e.g. NotePracticeMode).
- MainActivity sets currentMode in Model to NotePracticeMode.
- Model notifies Controller that it has been changed and passes currentMode.
- Controller then set currentQuestion in Model and tells Model to change currentFragment given currentMode in Model.
- Controller also tells FragmentManager to perform changing Fragment in view.
- Controller then continues Scenario 2

Scenario 4
- NoteFragment listens to FilterButton in view
- User clicks the FilterButton
- NoteFragment then tells MainActivity to start NoteModeFilterPageActivity and expects a result.
- NoteModeFilterPageActivity gets started then listens to the four spinners.
- After a spinner is triggered to select an item by user, NoteModeFilterPageActivity apply current filters to get an array of generated notes.

- NoteModeFilterPageActivity also listens to the BackButton User clicks on BackButton NoteModeFilterPageActivity passes the array of notes as array of integers to MainActivity as an event.
- MainActivity handles intents by telling controller sets the current note pool in current question in model owned by MainActivity. Controller then continues Scenario 2

2.C Design Patters

Factory Pattern

- Question and GeneralFragment both use this pattern to generate specific type of Question or GeneralFragment (e.g NoteQuestion, IntervalFragment) given a Mode.
- There is already an inheritance relationship between Question, GeneralFragment and their children. This pattern enables us to create a child only provided a Mode instead of using a switch statement which we did before.

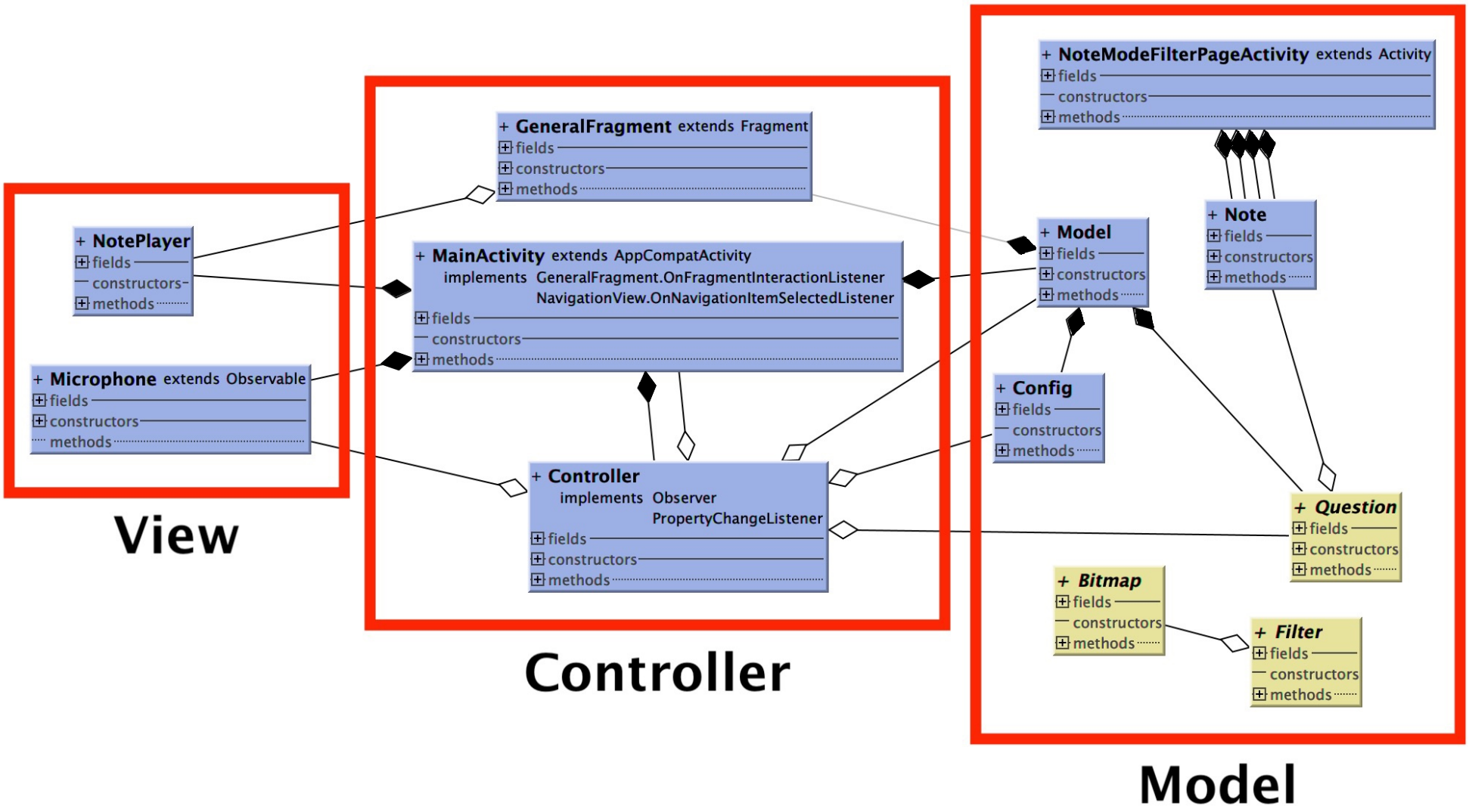
Observer Pattern

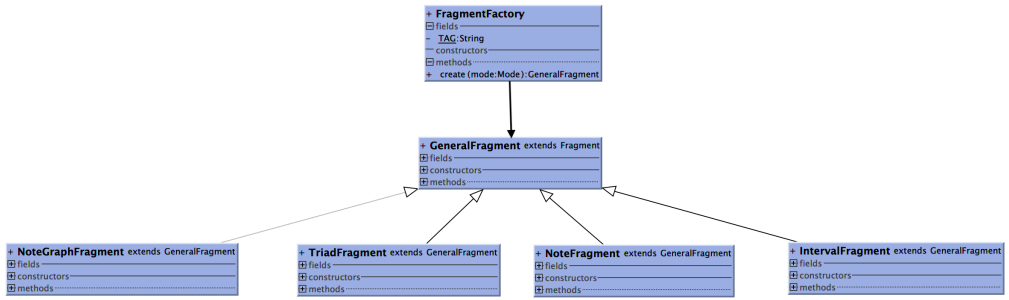
- Controller observes Microphones inner buffer changes so it can take action (e.g. to update view) when there is a frequency change event.
- Controller is also an observer for Model, where whenever a property in Model changes, it gets informed and take action (e.g. currentMode is changed by MainActivity in Model then Controller gets notified and perform changing currentFragment in Model).
- When MainActivity changes currentMode in Model, if Model doesnt automatically notifies Controller, MainActivity needs to perform changing currentFragment by itself (and other things it needs to do), which is easy to forget. After adapting observer pattern, all business logic of changing currentFragment in Model and other things to do all belong to Controller which makes perfect sense.

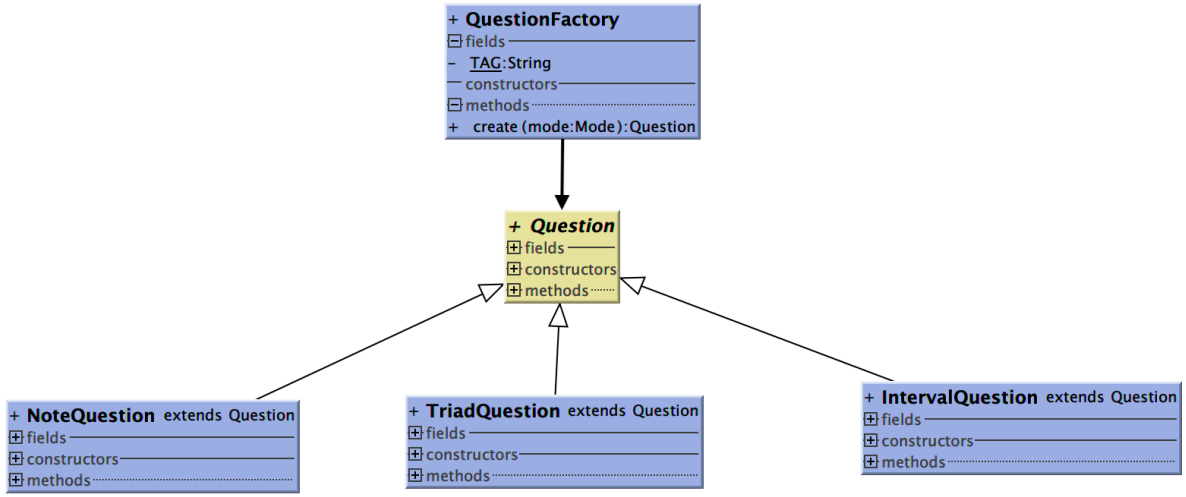
2.D Coupling Minimizing and Future Development

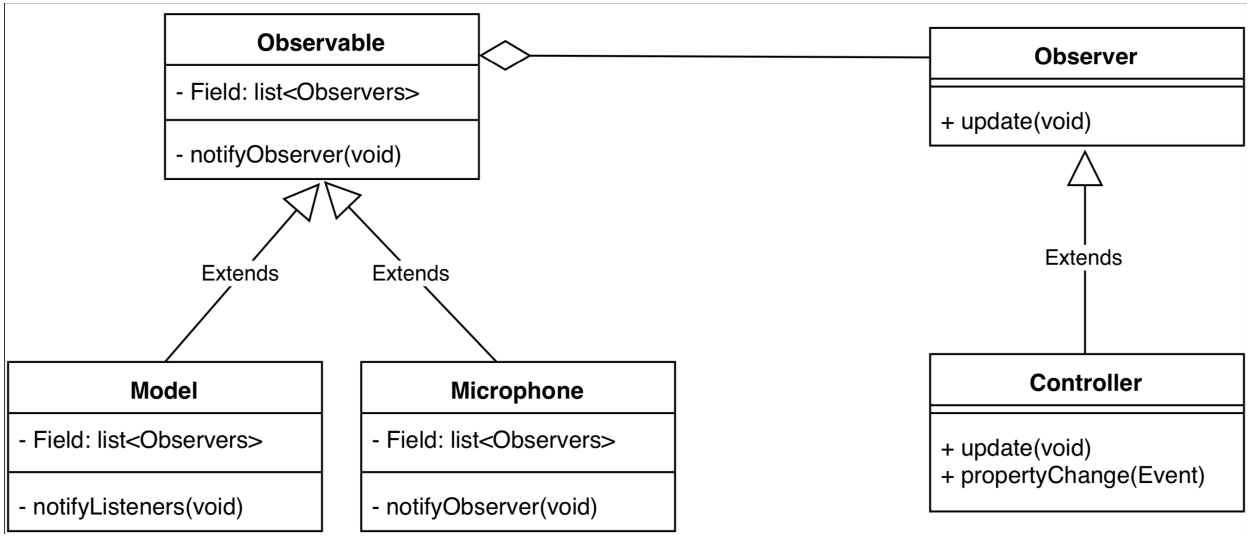
For our program we have some data coupling as a consequence of the MVC architecture, the model always passes data to the main activity, and the main activity to the active fragment. There is no data in which is processed later being passed around. Our app has sequential cohesion, meaning that it has data passing through the parts but no data is operated by two things at once. Again, the model controller is the only one doing calculations or the questions, but never both at once.

In the future we would definitely need to add more modes, specifically the song mode. As explained above we have made the system easy to generalize - to add modes we can simply add a new type of fragment that extends the GeneralFragment and add it to the navigation. We can then add the logic to handle the backend for it easily and pass data through to the fragment with a callback.


+ NotePlayer
fields
constructors
methods
+ Microphone extends Observable
fields
constructors
methods
View
+ GeneralFragment extends Fragment
fields
constructors
methods
+ MainActivity extends AppCompatActivity
implements GeneralFragment.OnFragmentInteractionListener
NavigationView.OnNavigationItemSelectedListener
fields
constructors
methods
+ Controller
implements Observer
PropertyChangeListener
fields
constructors
methods
Controller
+ NoteModeFilterPageActivity extends Activity
fields
constructors
methods
+ Model
fields
constructors
methods
+ Note
fields
constructors
methods
+ Config
fields
constructors
methods
+ Question
fields
constructors
methods
+ Bitmap
fields
constructors
methods
+ Filter
fields
constructors
methods
Model


+ FragmentFactory
fields
- TAG:String
constructors
methods
+ create (mode:Mode):GeneralFragment
+ GeneralFragment extends Fragment
fields
constructors
methods
+ NoteGraphFragment extends GeneralFragment
fields
constructors
methods
+ TriadFragment extends GeneralFragment
fields
constructors
methods
+ NoteFragment extends GeneralFragment
fields
constructors
methods
+ IntervalFragment extends GeneralFragment
fields
constructors
methods


+ QuestionFactory
fields
- TAG:String
constructors
methods
+ create (mode:Mode):Question
+ Question
fields
constructors
methods
+ NoteQuestion extends Question
fields
constructors
methods
+ TriadQuestion extends Question
fields
constructors
methods
+ IntervalQuestion extends Question
fields
constructors
methods


Observable
- Field: list<Observers>
- notifyObserver(void)
Observer
+ update(void)
Extends
Extends
Extends
Model
- Field: list<Observers>
- notifyListeners(void)
Microphone
- Field: list<Observers>
- notifyObserver(void)
Controller
+ update(void)
+ propertyChange(Event)

2.E Work Breakdown

Classes:

Alex: Bitmap, Config, Filter, FilterHandler, GeneralFragment, Interval, IntervalFragment, IntervalQuestion, IntervalsBitmap, IntervalsFilter, MainActivity, Mode, ModelController, Note, NoteFragment, NoteGraphFragment, NoteModeFilterPageActivity, NotePlayer, NoteQuestion NotesBitmap, NotesFilter, NotesRangeFilter, NotesScale, NotesScaleFilter, OffTrackLevel, Question, Triad, TriadFragment, TriadQuestion, VoiceListener, Layouts

Jia-lin:GeneralFragment, IntervalFragment, IntervalsBitmap, IntervalsFilter, MainActivity, ModelController, Note, NoteFragment, NoteGraphFragment, NoteModeFilterPageActivity, NotePlayer, NoteQuestion, NotesFilter, NotesRangeFilter, NotesScale, NotesScaleFilter, OffTrackLevel, Question, Triad, TriadFragment, TriadQuestion, VoiceListener, Layouts

Johnny: MainActivity, ModelController, Note, NoteFragment, NoteGraphFragment, NoteModeFilterPageActivity, NotePlayer, NoteQuestion, NotesFilter, NotesRangeFilter, NotesScale, NotesScaleFilter, OffTrackLevel, Question, TriadFragment, NoteModeFilterPageActivity, TriadQuestion, Layouts, GeneralFragment, IntervalsBitmap

Irvin: GeneralFragment, IntervalFragment, MainActivity, ModelController, Note, NoteFragment, NoteModeFilterPageActivity, NotePlayer, NoteQuestion, NotesFilter, NotesRangeFilter, NotesScale, NotesScaleFilter, OffTrackLevel, Question, TriadFragment, TriadQuestion, VoiceListener, Layouts, GeneralFragment, IntervalFragment,

Architecture: Model: Alex, Jialin, Johnny, Irvin View: Alex, Jialin, Johnny, Irvin Controller: Alex, Jialin, Johnny, Irvin