Are You Kitty Me -- Design Manual

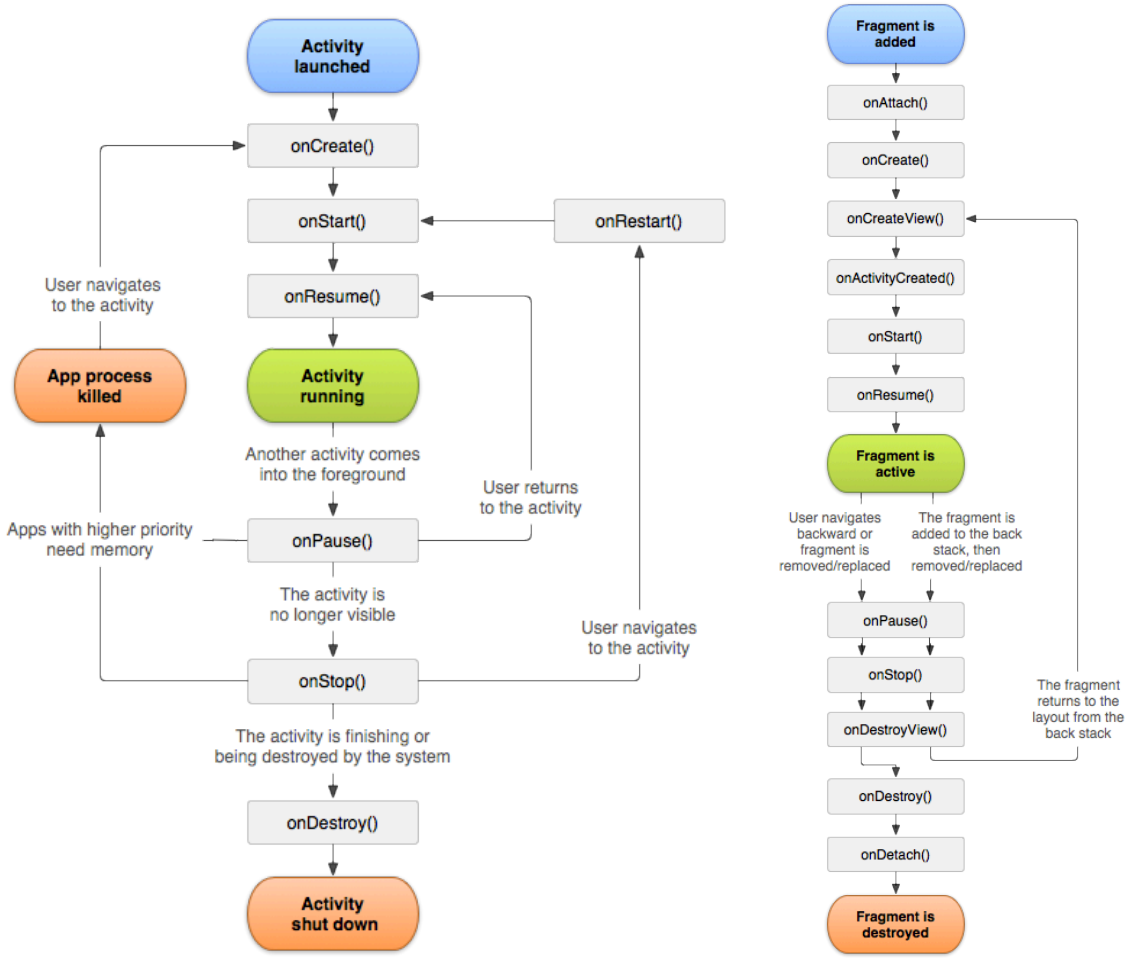
By Cade Chen, Jingya Wu, Haoyu Xiong, Sarah Xu

Introduction

For the CSCI 205 Final Project, our team built a multi-purpose app combining productivity, fitness, as well as vocab studying, that integrates a cat feeding system to motivate the user to complete his/her goals on a daily basis. To achieve our purposes, we decided to build an Android app using Android Studio, as Google provides plenty libraries and frameworks that, to a greater extend, facilitate the process of delivering functionalities to our customers.

An Android application is always made of Activities and Fragments. By definition, a fragment is a modular section of an activity. In other words, fragments are contained within activities, but they perform completely separate behaviors in their own lifecycles (explained below). The Activity class is a crucial component of an Android app. It serves as the platform on which user interacts with the app, and also provides the window in which the app draws its UI. To put this more straightforwardly, activities are simply the different pages of interfaces of the app. When the user navigates among different pages, he/she is usually navigating from one activity to another. Both activities and fragments are created and implemented through their lifecycle methods. These methods are either triggered by the OS or by another activity/fragment. For instance, when an application is first opened, the onCreate method of the first activity will be triggered; and when an activity is no longer visible, the onStop method of the activity will be called. Two diagrams illustrating the relationship among lifecycle methods of activity and fragment are shown on the next page.

Now we have an understanding of the fundamentals of our Android app, let's take a look at the architectural design of our app. In our Are You Kitty Me app, we followed the Model-View-Controller (MVC) design pattern to maintain a clean and readable code base. The model for our app is the User class that is independent from all activities and view. The main purpose of this class is to store the current state of the system (cat and user's stats). Our model stores cat's attributes (mood, health, and wealth) and user's daily goals, and can be updated by the controller classes, which are, in our case, the activities. Activities controls the state of the model and modifies the model as user interacts with our app. For instance, when the cat's name is changed in the settings activity, our User class will be notified to change the name attribute, and that leads to the final stage of our program structure -- the view. Similar to SceneBuilder of JavaFX, android provides a simple drag-and-drop tool for GUI design, which then automatically generate xml files for us. These xml files form the view of our app. View is directly seen by the user and is constantly updated by the controller (activities) via the unique ID that is assigned to each layout element by the developer.



Are You Kitty Me also employs Service to perform background tasks when the application is not up and running. Service is an application component that can perform background operations and that neither provides nor requires a user interface. For instance, the model (User class) needs to be reset at the start of each day in order to keep track of the user's daily progress. This is then achieved by implementing a Service class that is triggered by AlarmManager everyday at midnight. In addition, our app also uses SQL database to store vocabs and the learning progress. We will discuss it in more detail in the vocab section below. All in all, our app integrated Activity, Fragment, Service, and many other handful tools and libraries.

User Stories

I want ...

1. To have a lovely cat to raise
2. To feed the cat food
3. Something to motivate me to walk more everyday

4. Something to remind me to study vocab
5. Something to prevent me from using my phone
6. A reward system in the app that demonstrate my achievements
7. A store to buy food and other products
8. To set customized daily goals
9. An efficient vocab learning tool that applies learning curve

Are You Kitty Me currently services the following features: cat adoption and feeding, grocery store for cat's food, vocab learning tool that implements Ebbinghaus' Forgetting Curve, and a Pomodoro timer. User stories 1 and 2 are accomplished by our adopt page and main page, where the user can adopt and name a cat, look at a lovely animated cat, and feed the cat food by long-clicking the cat's animation. Our vocab learning feature that implementing the Forgetting Curve fulfills user stories 4 and 9; the Pomodoro timer that block the user from accessing other applications on the phone and the grocery store that sells cat's food accomplish user stories 5 and 7 respectively. In addition to these, our app also provides a Stats page that keeps track of and displays the user's step count and other progress (user stories 3 and 6), as well as a Settings page that the user can use to customize his/her daily goals (user story 8).

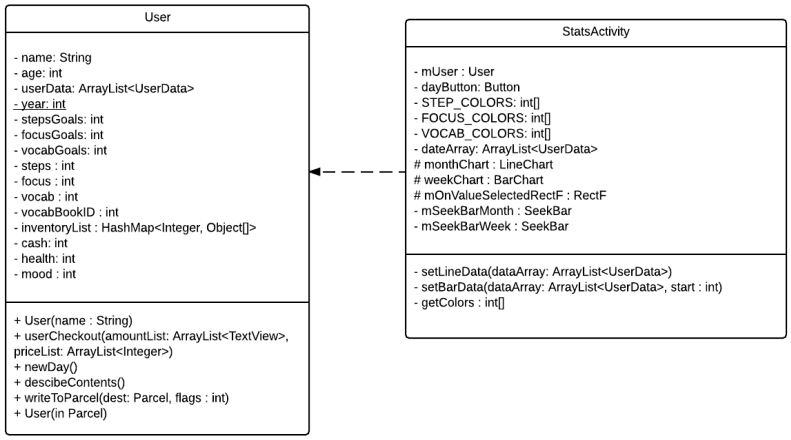
OOD

Overview

As mentioned previously, Are You Kitty Me implemented the traditional Model-View-Controller (MVC) design pattern. Our team chose to use this pattern over MVP (model-view-presenter) because MVC does not require us to break down each activity into view and presenter. Instead, our activities serve as the controller, and the User class and xml layout files serve as the model and view of our design pattern. However, one concern of ours about using this design pattern is that our controller (activities) contains part of the view component since we have to manipulate and generate non-persistent layout elements in the activities. We finally decided to live with this design pattern, as it is the most familiar design pattern to us and is the clearest that we can implement for our app.

One of the challenges that we faced while building the app is how to pass our model around different activities. Unlike a project written in traditional Java, our controller classes (activities) do not allow object to be passed in from constructor. Instead, android app progresses from one activity to another through its lifecycle methods. Hence, we implemented parcelable in the User class so that we can add the User object as extended data to intent by *putExtra*. By doing so, we are able to keep updating our model without reinitializing it every time the front activity is changed.

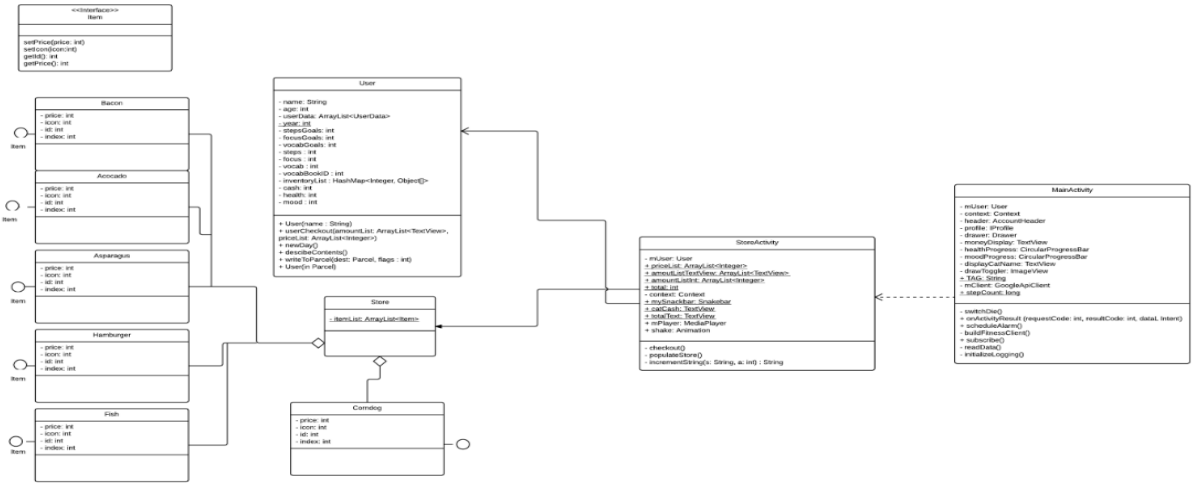
Stats



The model for STATS activities is the User class which receives and keeps track of data from other activities. It displays data generated by Vocab activity, Timer activity and Google Fit API. The User object created during Adopt activity is passed into Stats activity as parcelable and userdata retrieved is visualized with PhilJay library. Since the Stats activity does not modify any user values, we do not need the main activity to wait for results from the stats intent. The Stats activity, acting as controller in the MVC model, communicate with its view class implemented in xml. Since the layout of statistics is relatively simple, we utilized linear layout format to arrange the charts and buttons.

Note, in order to display the functionality of the statistics activity, we generated pseudo data to simulate user's daily progress. Before actually employ the app to market, the stats activity can be easily modified to reflect real data.

Store

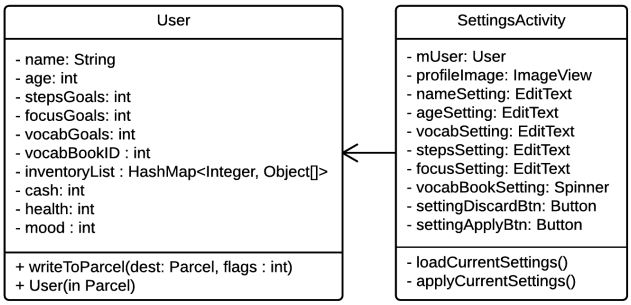


The model here is quite simple. The Store class when first initiated, creates a list and puts all the items into the list. The food items all implements Item interface. All the items contain four attributes: price, icon, id, and index. The index is given to the food item on creation. The index will determine how and where the item will be presented.

Most of the view in *StoreActivity* is inserted by code. Doing so will allow us to insert an arbitrary number of items into the view.The view provided by the xml file is only a stationary bar at the very bottom of the screen.

The controller is using *StoreActivity*. It handles the transaction and stores the purchase items' information into the user.

Settings



The Model of setting is the User class, which is created and initialized in the *AdoptActivity* and passed in to SettingsActivity as a parcelable object. The *mUser* object is modified by the SettingsActivity controller when the APPLY button is clicked. When back button is clicked, the controller (SettingsActivity) will pass the modified model (mUser) back to the MainActivity to display the updated User attributes on the main page.

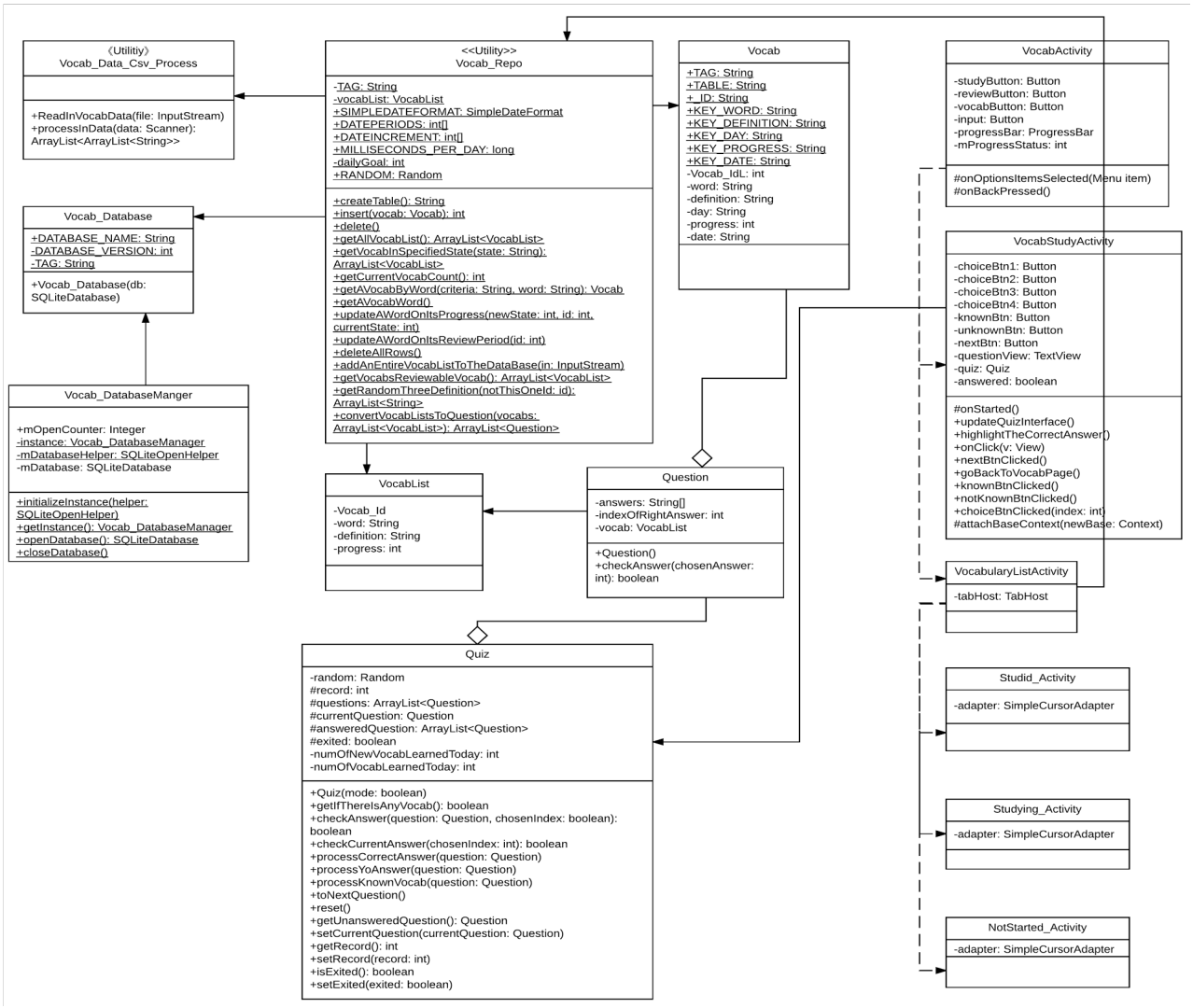
Vocab

Models

VocabActivity itself doesn't have a model since it's more like a navigation menu that leads to different features. *VocabStudyActivity*'s main model class is the Quiz. The quiz class provided the activity with Questions based on Question class which has the word and four corresponding options, one of which is true. The Activity, with the user choosing between different options (including know, don't know and next), will call *Quiz*'s methods accordingly.

However the *Quiz* class has connections much further in the back end. As stated above, Quiz is based on an array of Question class which also include *VocabList* class. *VocabList* class is an medium class used for processing between database and the model. *VocabRepo* class is an utility class designated solely for interacting with database. The quiz will class almost every method(for example, when the user got right for a question, activity class will call quizzes' processCorrectAnswer class, which subsequently

call updateAWordOnItsProgress or updateAWordOnItsReviewPeriod methods). Vocab_Database extends from SQLiteHelper to manage the database directly. However, due to various inconveniences, we decided to employ a Vocab_DatabaseManager class to better manage the database with synchronized design.



VocabularyListActivity's model is the utility class Vocab_Repo because it's displaying the words based on their progress. Since we are utilizing the tabhost, we created three different fragment activity corresponding to the progress.

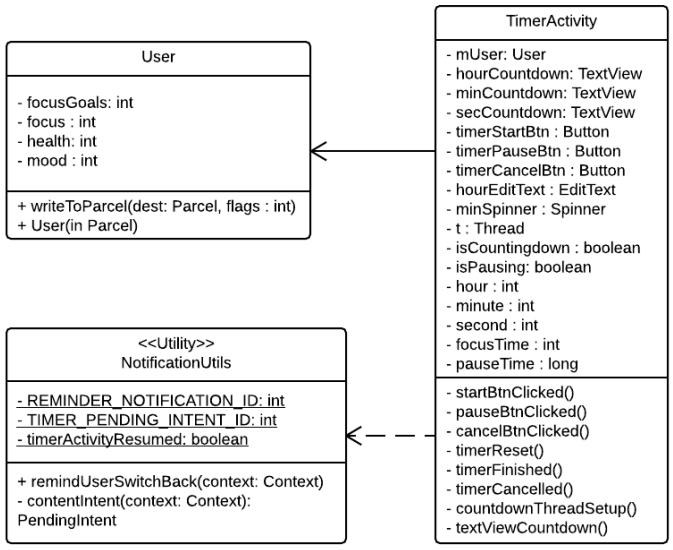
Database

Vocab_Repo class is basically the library we implemented for vocabulary management since we want to keep track of the progress and comprehension level reflected by Ebbinghaus's forgetting curve. Every method in the repo class serves for a specific purpose. As did a lot of android developer, we used cursor to parse and filter out the informations we want.

Controller

The right hand side of the above graph denote all the activity classes (controller). VocabActivity handles the navigation between user and various other activities within the vocab feature. VocabStudyActivties handles the learning process through the use of Quiz. VocabularyListActivity aims at displaying the content (vocab list) on the view.

Timer



The Model of timer is the User class, which is created and initialized in the *AdoptActivity* and passed in to *TimerActivity* as a parcelable object. The *mUser* object (focus, health, and mood attributes) is modified by the *TimerActivity* controller when a focus task is completed or cancelled. *TimerActivity* depends on the *NotificationUtils* class which handles the push notification that is triggered when user switches out of *TimerActivity*. When back button is clicked, the controller (*TimerActivity*) will pass the modified model (*mUser*) back to the *MainActivity* to display the updated User attributes on the main page.

Conclusion

Overall, this project was both fun and educational. Throughout the development process, we started with essentially zero knowledge with Android Dev and grow accustomed to Android jargons such as activity, intents, content provider, lifecycle and so on. As to the graphical design part, we strived to closely follow the material design practices. To get started with Android development, we get off the ground by following Google's Android Basics Course on Udacity. Since we were all novices, we resorted a lot of external references in developing our app. Below is a list of the sources cited:

1. Android tutorial: https://www.udacity.com/course/new-android-fundamentals--ud851
2. Google Fit for Android: Reading Sensor Data: https://code.tutsplus.com/tutorials/google-fit-for-android-reading-sensor-data--cms-25723

3. inputstream to string
   http://stackoverflow.com/questions/309424/read-convert-an-inputstream-to-a-string
4. Multi threading with **AsyncTask**
   http://developer.android.youdaxue.com/reference/android/os/AsyncTask.html
5. Add back button
   http://stackoverflow.com/questions/34110565/how-to-add-back-button-on-actionbar-in-android-studio
6. Color scheme
   http://www.creativecolorschemes.com/resources/free-color-schemes/cute-color-scheme.shtml
7. DragNDrop
   http://www.vogella.com/tutorials/AndroidDragAndDrop/article.html
8. Swipe Tab navigation
   https://developer.android.com/training/implementing-navigation/lateral.html
9. Icons Library
   https://github.com/mikepenz/Android-Iconics
10. Plotting Library
    https://github.com/PhilJay/MPAndroidChart
11. Images
    https://developer.android.com/reference/android/app/Notification.BigPictureStyle.html
    http://www.tothenew.com/blog/android-notifications-with-images-and-text-together/
12. Thread and update time
    http://stackoverflow.com/questions/14814714/update-textview-every-second
13. Navigation drawer
    https://developer.android.com/training/implementing-navigation/nav-drawer.html#DrawerLayout
14. Navigation drawer library
    https://github.com/mikepenz/MaterialDrawer
15. Circular progress bar
    https://github.com/lopspower/CircularProgressBar
16. Service and AlarmManager
    https://guides.codepath.com/android/Starting-Background-Services
17. Espresso Testing
    1) Disabling Animations
    https://product.reverb.com/disabling-animations-in-espresso-for-android-testing-de17f7cf236f
    2) https://gist.github.com/xrigau/11284124
18. French, German, Spanish GCSE vocab spreadsheets
    https://www.tes.com/teaching-resource/mfl-vocabulary-spreadsheets-11281805
19. MVP vs. MVC
    https://www.techyourchance.com/mvp-mvc-android-1/
20. Convert object to JSON string then save in sharedPreferences
    http://stackoverflow.com/questions/7145606/how-android-sharedpreferences-save-store-object
21. Popup
    https://developer.android.com/training/snackbar/showing.html
22. ViewPager
    https://developer.android.com/training/animation/screen-slide.html#pagetransformer
23. Icons
    http://www.flaticon.com/search?word=arrow&license=all&color=0&stroke=0
24. Sound sources
    http://www.zapsplat.com/