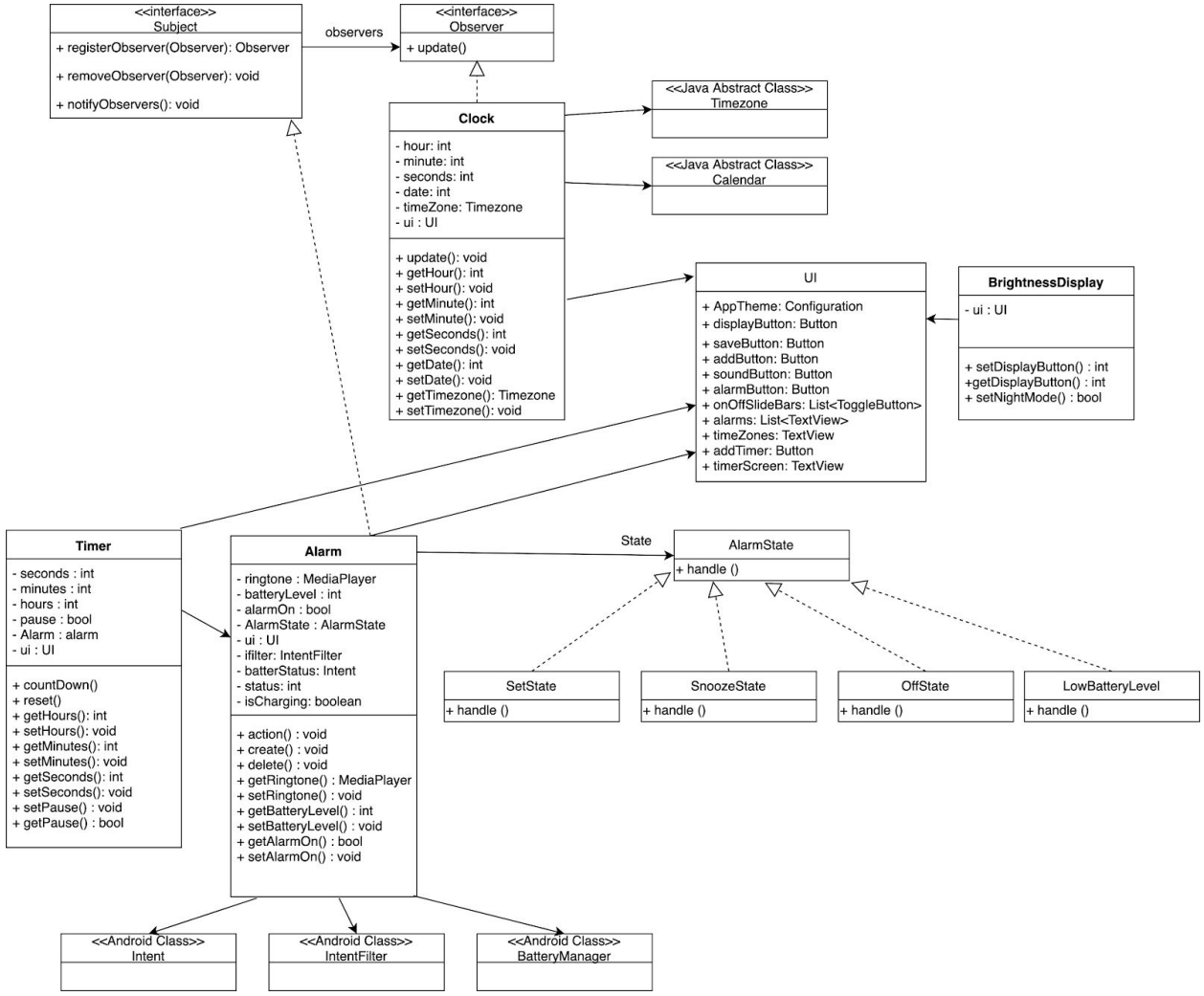
Project Title: Plug It In Stupid
Team Names: Aaron Steiner, Parikshit Bhetwal, Kyle Schultz

Final State

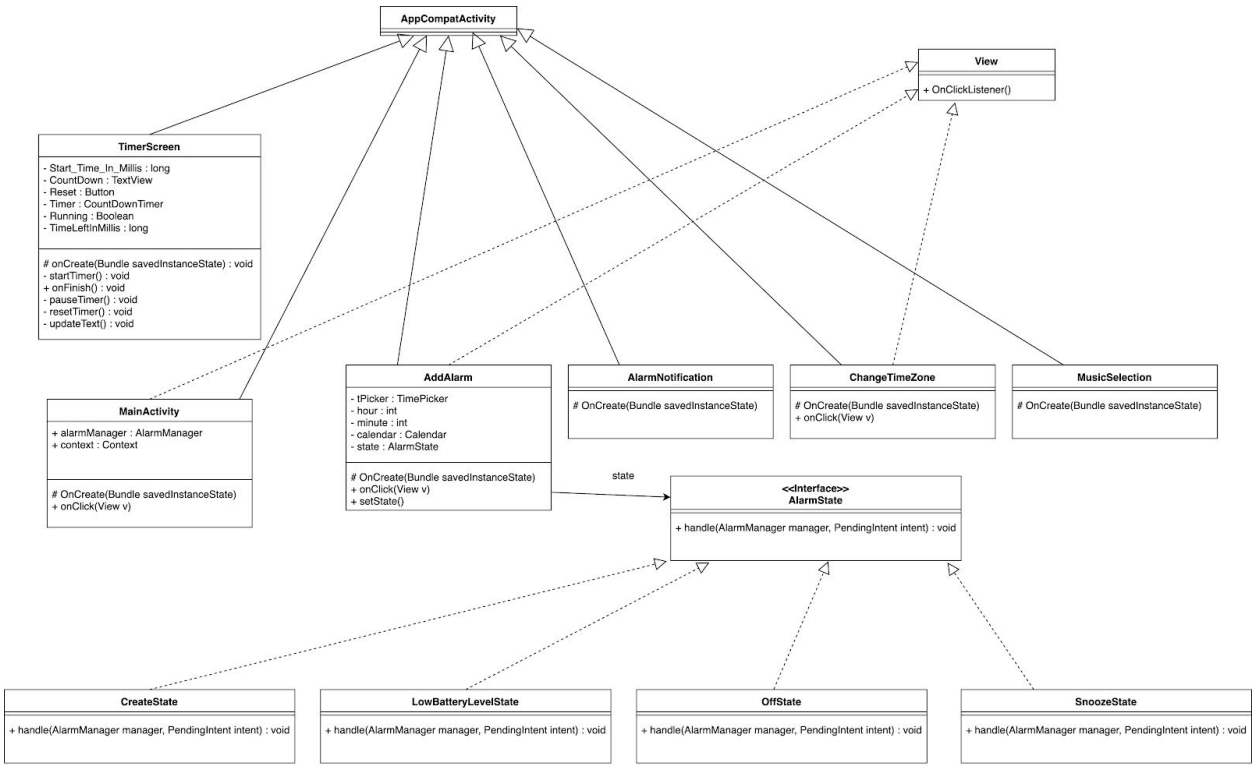
The final state of the system is that it is fully functional. We implemented all of the features that we said we would. A user can select a ringtone from the music selection screen and their chosen song will be used for the alarm. A user can change the display of the app to be in dark mode for night time and back to light mode for daytime or whenever they wish. A user can select whatever timezone they wish to use and the app will set the alarm off at the appropriate corresponding time in the timezone (i.e. if it's 8am MST, the user selects EST and sets an alarm for 10am, it will go off). A user can add an alarm, select the time they want it to go off, pick what days of the week they want it to repeat, or cancel the addition. The user can also toggle the alarm on and off. When the alarm goes off the user can choose ok and the alarm will be set for the same time 24 hours from then, unless switched off. The user can also select snooze and in this case, the alarm will remain off for the next ten minutes and then will go off again. If the user adds more alarms than there is screen space, they can then scroll down to see their other alarms. If the battery falls below 15% and the phone is not currently charging, the app will send a notification to plug in the phone so that they won't miss their alarm due to the phone dying. The only feature that isn't exactly how we said it would be implemented is the timer. Instead of letting the user add any length timer that they want, they can only use a 10 minute timer. During anytime the timer is running they can pause, resume, and reset the timer. We decided to not implement the user editable timer because we had to focus on other class projects and didn't have the time to figure out getting the inputted value and using it for the timer. We also did not implement the LowBatteryState because the BatteryService did this functionality for us while running in the background of the application.

Final Class Diagram and Comparison

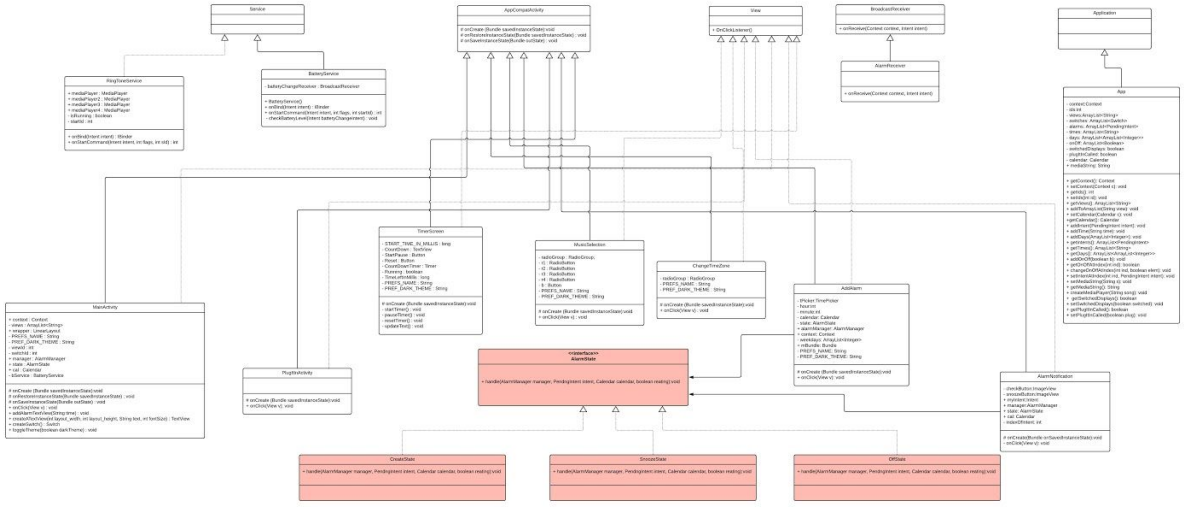
Project 4 UML Diagram



Project 5 UML Diagram



Project 6 UML Diagram



The classes that are highlighted in light red are classes that are using the state pattern. See:https://www.lucidchart.com/documents/view/e491960b-f24c-47a0-8f7c-b46947a835d6/0_0 for a larger view of this UML diagram.

Key Changes from projects 4 and 5 to project 6:

One key change from the earlier projects to project 6 was understanding how to develop an Android application. This allowed us to be able to implement and extend the proper Android classes we needed to build the project.

Another key change was how monitoring the battery level and how it was related to the AlarmState. In android, there is a BatteryManager that can be used as a background service to monitor if the phone is charging and if it is at a certain battery level. There was no need for a LowBatteryState because if the battery was at the threshold we specified, then the Service would broadcast and go to the PlugItInStupid activity and start the RingtoneService.

Yet another change was figuring out that we needed to use services in our project. In order to have the ringtones play during the alarm we needed a service to run when this was called. We also needed a service to monitor the battery level.

Third-Party Code vs Original

Some of the functionality of the application is built into android. Many classes are provided to allow for creating UIs and starting activities for different functionalities of the app. Intent, PendingIntent, AlarmManager, and Services are all built into android and proved to be very useful to get our application functioning properly. Although these are built into android, we had to write a lot of custom code to get proper functionality. Also, many files are created when you create a new Android Studio application and we did not create these files (i.e. build.gradle).

Most of the code is original but many of the ideas came from researching the android developers website since there wasn't much pre-existing knowledge.
Link: https://developer.android.com/

The app timer was built off of a coding in flow tutorial.
Link: https://codinginflow.com/tutorials/android/countdowntimer/part-1-countdown-timer

The media player and ringtone service were figured out using a Github repository as a reference.
Link:https://github.com/annathehybrid/RichardDawkinsAlarmClock/tree/master/app/src/main/java/com/example/axu1/richarddawkinsalarmclock

To learn how to dynamically insert a TexView to the UI when a new Alarm is created we used this reference from StackOverflow

Link:https://stackoverflow.com/questions/11398103/how-to-use-addview-to-add-view-to-layout

To learn how to monitor battery level on an android phone we used this reference to know when the phone is at a low battery state:
Link:https://developer.android.com/training/monitoring-device-state/battery-monitoring

Implementing a dark-theme to light-theme toggle was based off:
Link:https://www.chrisblunt.com/android-toggling-your-apps-theme/

A list of Android timezone ID's:
Link:https://gist.github.com/arpit/1035596

Statement On OOAD Process

1. One key design process issue we faced as a team was trying to use the waterfall method like the projects were pushing us to use. Having to know everything for documentation purposes before the app was even constructed without us knowing much about android development was difficult. We didn't know how the UML diagram and sequence diagrams would look exactly. Also, having to specify patterns that we were going to use was difficult because we didn't really know how stuff was going to be implemented yet.

2. A key design process that worked well was identifying the system requirements by writing use cases. The use cases allowed for use to know the essential features of the system and how the user would interact with them. This allowed for us to design and implement the system in a way that allowed for the user to accomplish what was essential for the system. We used the use cases to know the flow of the operations the user would take to accomplish a task, which was easily translated into Android activities and services that we could implement.

3. Another key design process element that we utilized was test with development. We would write a section of code and immediately test it to see if it worked. This allowed us to not get carried away with writing tons of code and then trying to debug it all in one chunk. By testing it in small sections we were able to easily identify problem areas and fix them. This made the process of combining the small tested chunks to the overall code base super easy and enabled a smooth

integration. We used the android emulator of the phone to manually test each feature and UI.