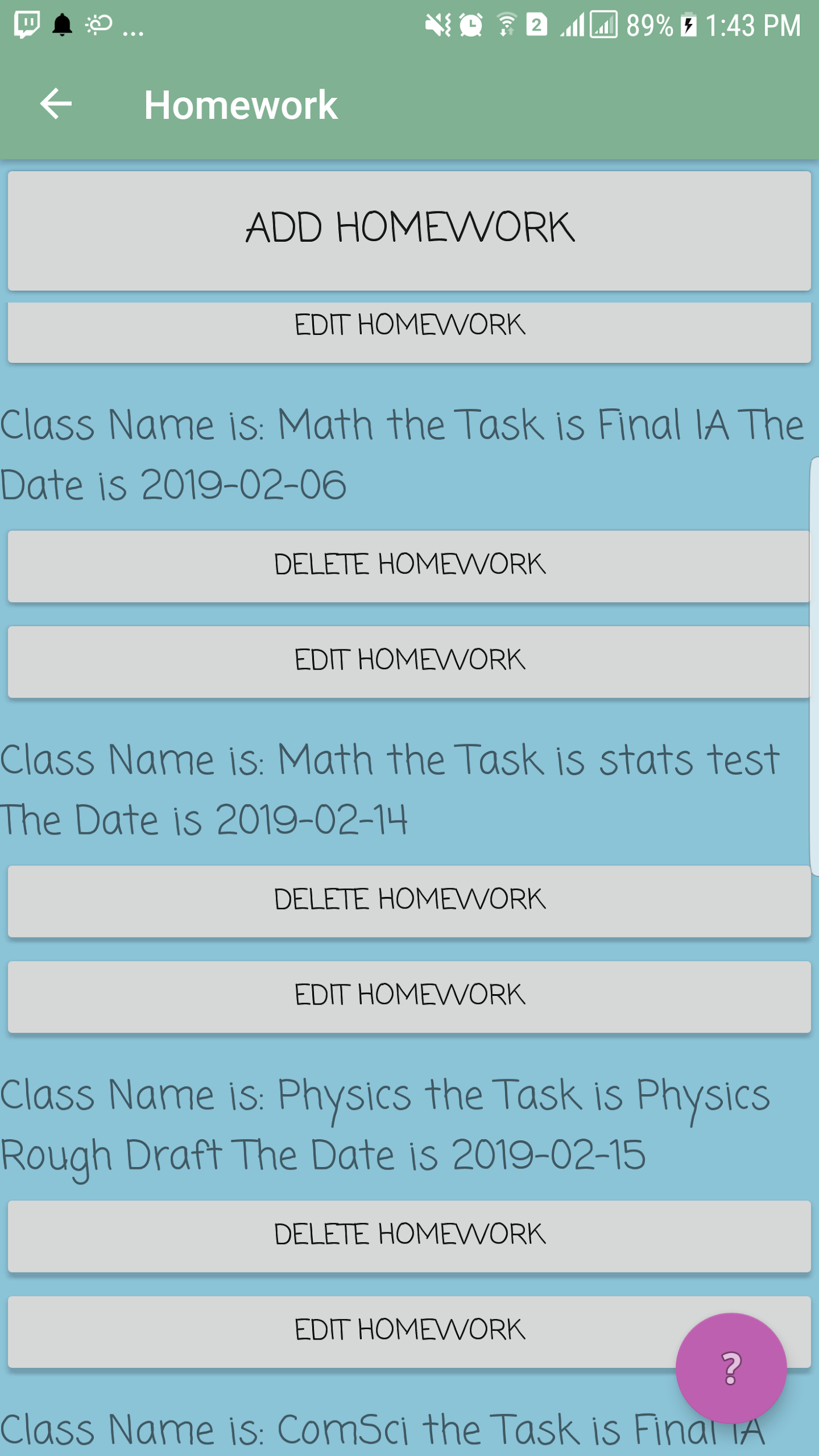
Criterion C: Development

**Complexity #1: Use of XML**





(The XML code for the floating button above)

(Visual Representation of the help button on right

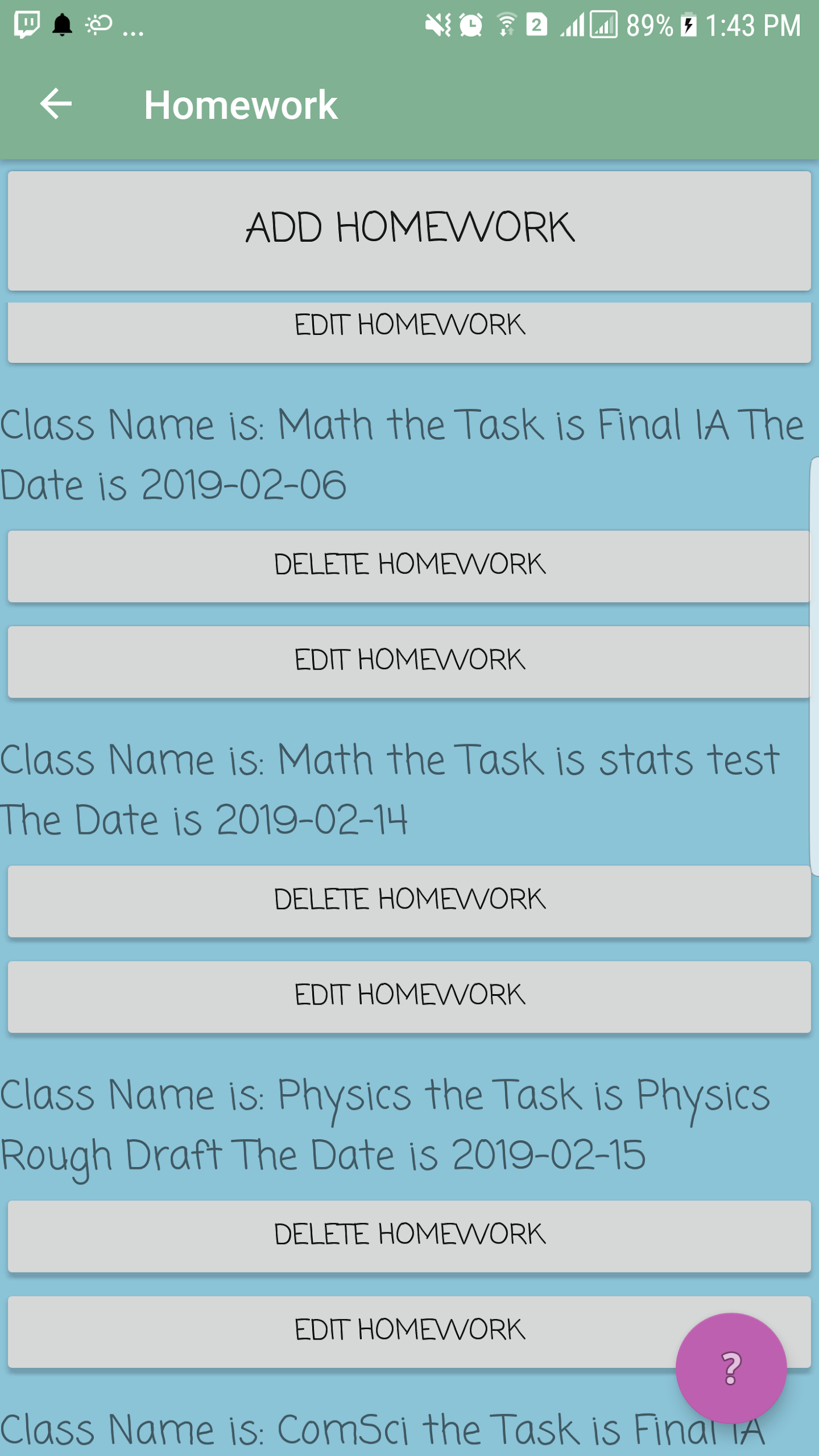
image)

**Problem:** Make a Help Button  
**Solution:** Floating Action Button (fab) via use of XML

**Rationale**: A fab is an XML feature and can be made by using XML classes. This solution allowed me to create a persistent help button that never moved, is floating and hence had no issues with managing different UIs. Because it never moved it was a perfect help button, once the user familiarizes with its location, it never changes which for an essential tool like help button is important. Fab also has no issues with UIs, it floats above and hence is very easy to program around.

(XML code for layouts in view classes)



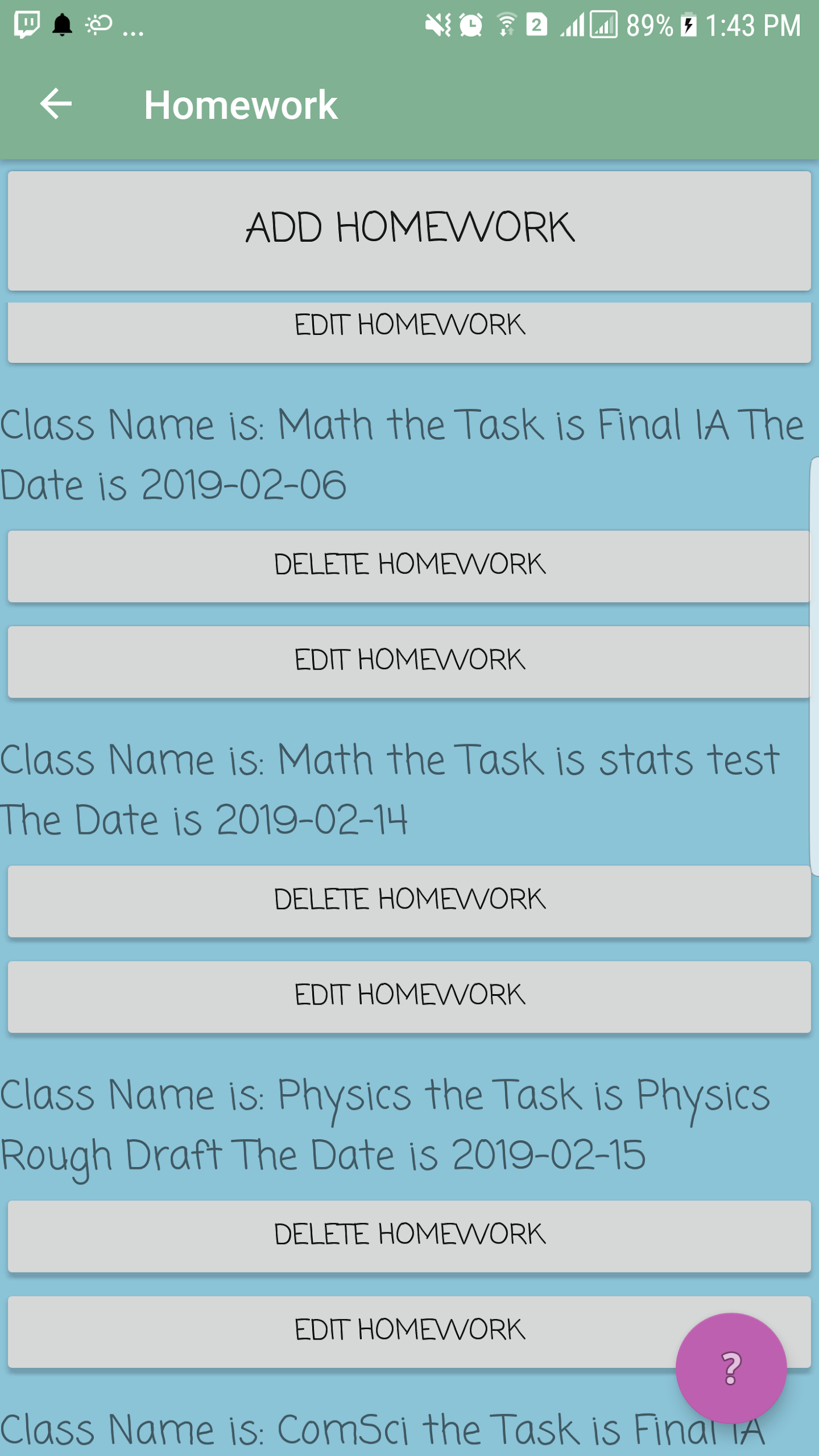
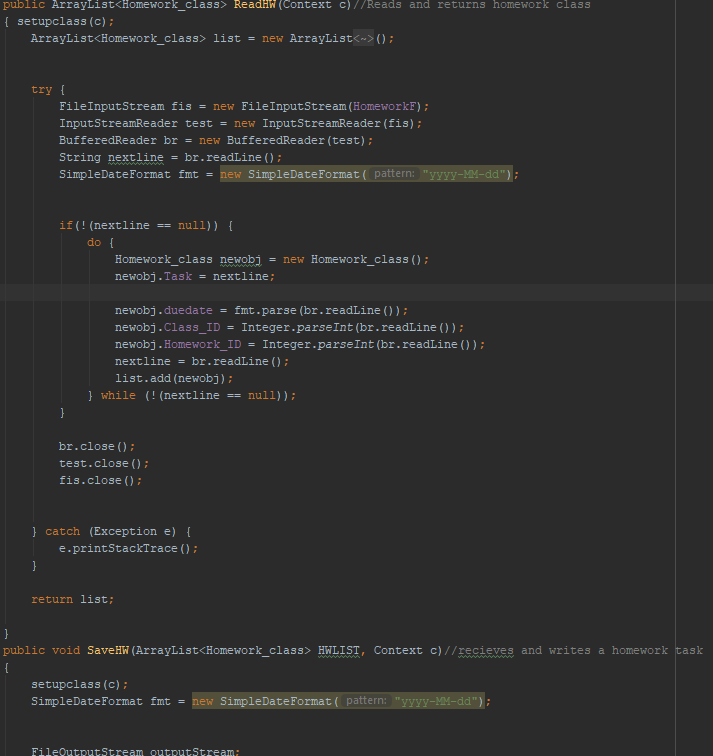
(GUI produced by that code)

**Problem**: Display all the homework according to the design made in criterion B

**Solution**: Use of Linear Layout in XML

**What it is and Why is it appropriate**: LinearLayout is a predefined XML layout that organizes XML items in a vertical or horizontal (for this use, the list is vertical). The Layout organized all the homework in a list but most of all, XML essentially acted as a go-between the Layout and backend code by allowing the Layout to be referred to in code, hence allowing the backend code to manipulate the list and create a dynamic list. XML created a list that is vertical, is easy to work with on the backend side and fits the pre-set design of criterion B to display homework.

**Complexity #2 Read and write (R&W)**

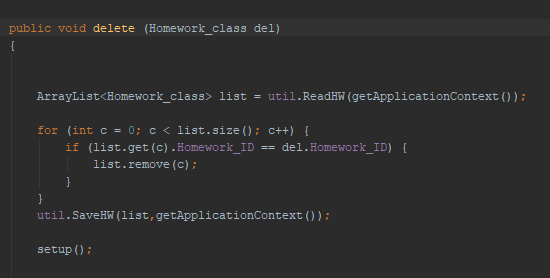


(Code for reading and writing Homework Class left)

(A use case of the methods on the right)

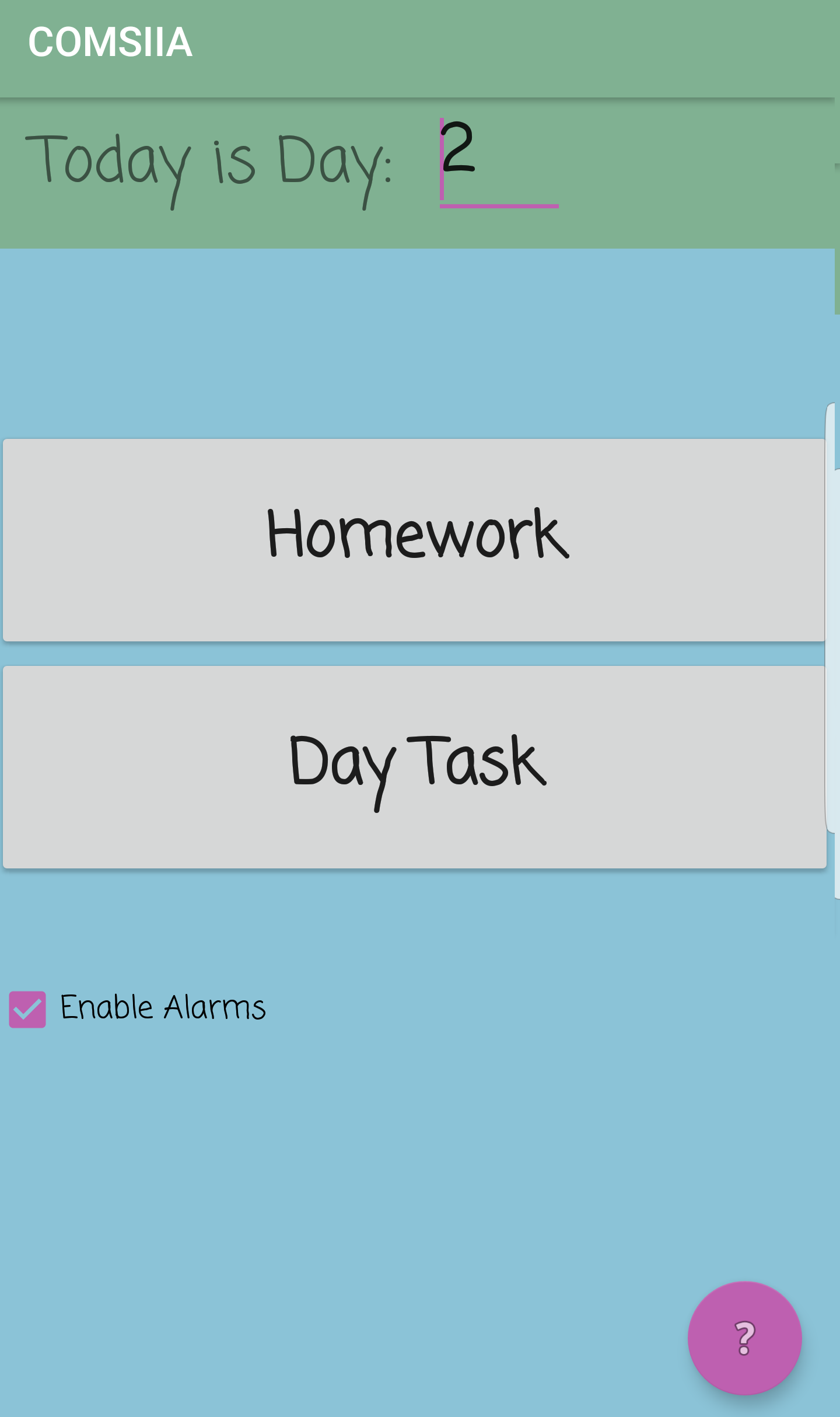
**Problem:** Manage multiple reads and writes with various purposes at many points in the program.

**Solution**: Create a class with methods that manage read and write of all files via Buffered Read and Write(R&W)

**What it is and Why is it appropriate**: Buffered R&W are a type of R&W that go through the entire file at every use rather than edit parts of it. The reason this solution was used was that R&W methods for all the files were stored in a separate class with their own methods. These methods are called throughout the program for various purposes. For example if a use case was to change a single item in the file, the entire file is used because if another use case needed to change multiple elements, creating multiple R&W methods would take too long to make and be unnecessary complex, simply using the entire file is preferable so it is easier to abstract the R&W methods in another class. Another reason to use buffered R&W is that the size of fields can vary for some fields and using RAF would increase the chance of errors and one of the most common use case for the R&W needed the entire file in any case.

The delete button is an example of such use. These functions (delete, edit, display) use the entire files to perform their functions. Finding the exact place where it was written and editing that section would have a high chance of failure, so the buffered R&W the entire file from start to finish to minimize risk.

**Complexity #3 Use of advanced codes and Algorithmic Thinking**



(Screen 1 of the application, note the checkmark instead of a button)

**Problem:** Setup repeating alarms that depend on user’s classes, School ‘Day’ and the day of Week

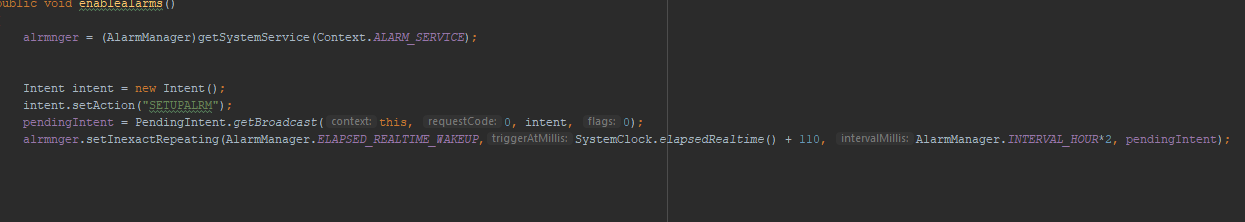
**Solution:** Use of exact, inexact repeating alarms and intents in a complex algorithm

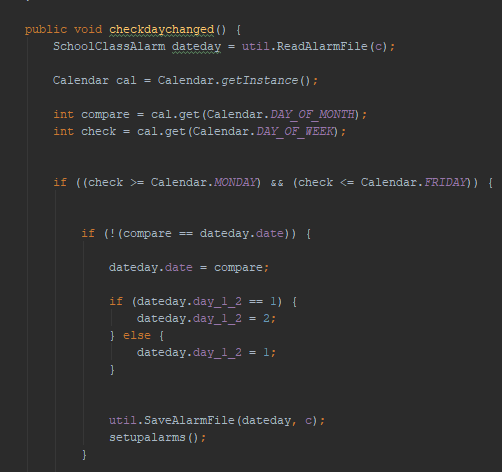
**What it is and Why is it appropriate**:

Exact Alarms - Internal Alarms that do a pre-set intent at a set time.

Inexact Repeating Alarms - Repeating alarms that go on a regular interval doing preset intents.

Intents - A action such as opening a new activity (GUI page) or broadcasting a system-wide message to be picked up by receiving classes

When the program is open, it creates a regular 2-hour inexact repeating alarm throughout the entire phone from phone’s first boot. This alarm indicates a method to check if the day of the month has changed, and hence should new alarms be set. This alarm runs every 2 hours to prevent chances of missed alarms. A sample use case would be if the application is opened and between 12 am to 2 am the alarm would trigger once and the receiver class would then manage the alarms accordingly. Any more of an interval than 2 hours would increase chances of missing too much if the phone is restarted at awkward times and any shorter would create unneeded overhead. 



The code on the left is part of the receiving class. This method deals with verifying if alarms should be set and further actions. It verifies that it isn't a weekend and the day of the month is different. If the day has changed, to change the current value and save and finally run the methods to set up the alarm. The importance of this algorithm being so short and efficient is because it runs many times and hence it should require minimal checks and variables created. So, the moment all the checks are done, new values of SchoolClassAlarm (custom datatype for alarm details, refer to appendix Source Code Page ) are saved and alarms are set. All the checks are done to ensure that self-automation is properly done since there is no school on the weekends and if the day of the month is different, it must be a new day. It does not need to check if it is the next day since that requires additional complications, but only if the value for the day of month changed. This is so it is easier to manage between changes like 30th to 1st, 31st to 1st and more. It could be replaced with day of the week but it makes no significant difference as both are int.

Words: 1000