Project Title: Herd Safety

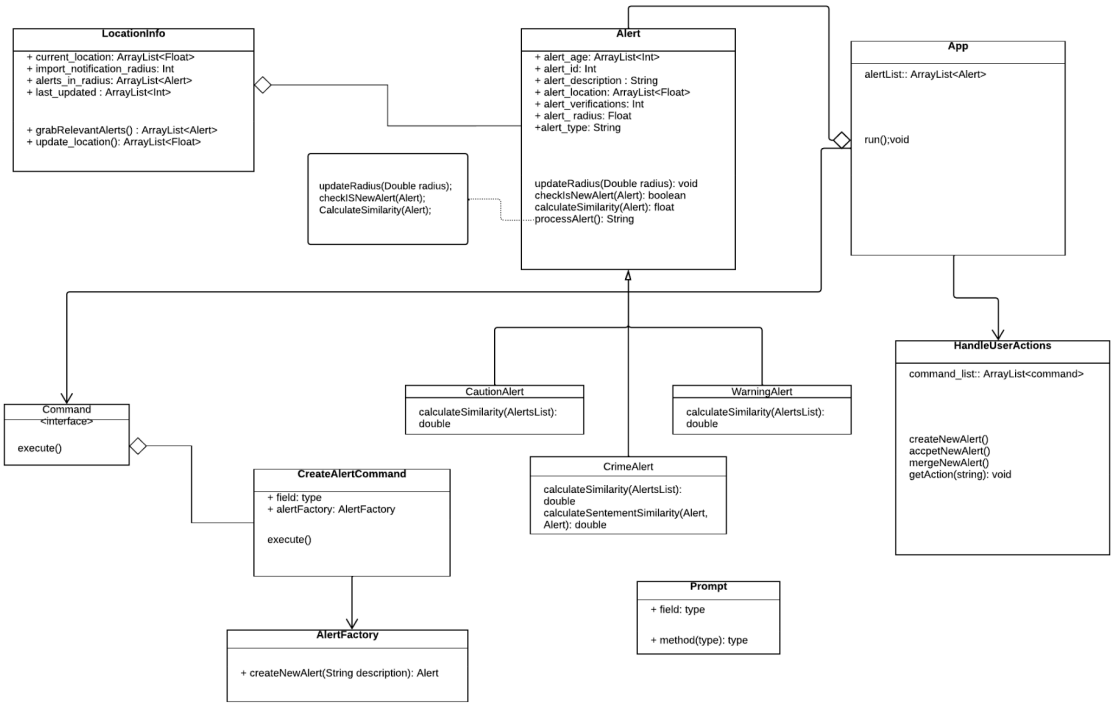
Team Members: Nova White, Kaleb Moore, Joseph Rizzo

Final State of System Statement:

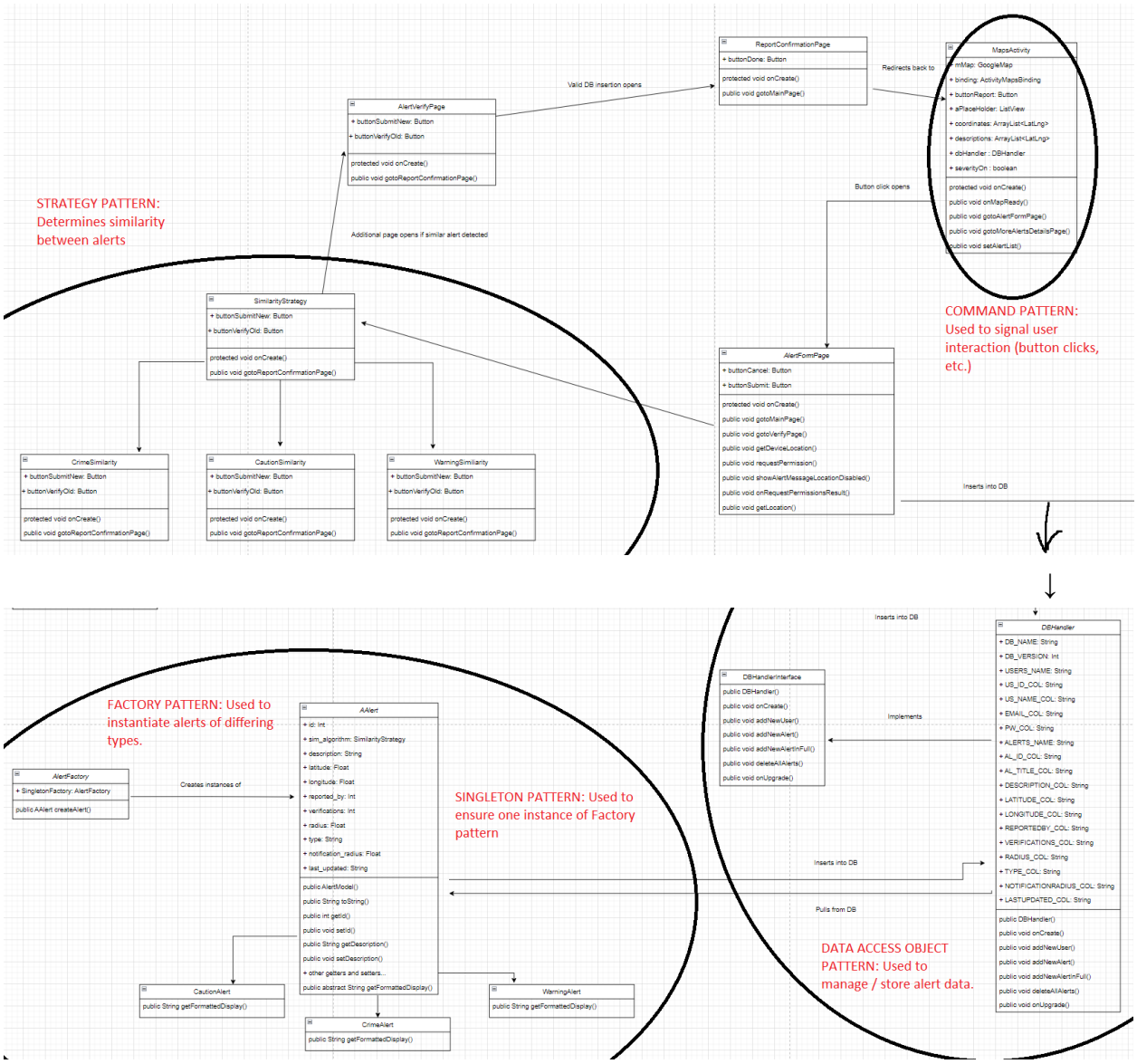
Our app is designed to alert the students of Boulder to all sorts of safety related events happening around the city! From the app, users can report public safety concerns on an interactive map interface, alerting others that are using the app. The alerts will be separated into 3 categories: Caution, Warning, and Crime. One can scroll around the map, clicking on interactive pins to examine further details about an event happening nearby, and can also view a chronological feed of Boulder's most pressing matters. The app would avoid false reporting by validating alerts that have had multiple users attempt to post the same information. If a user's post is similar to an existing post (within a certain time-frame) the app will ask to confirm with the user if they are reporting on a situation that already exists. This will decrease duplicate posts and simultaneously help credit existing posts. With all that said, the only thing that we had to change from our proposal to this point is our database tool. In our initial project proposal, we were discussing whether to use MySQL or PostgreSQL to implement our database, and we ended up choosing PostgreSQL to begin. However, upon further digging, we realized that we had overlooked how Android Studio comes with built-in SQLite functionality. For the scope of this project, we decided that it would be more straightforward to switch to SQLite to implement our database.

Final Class Diagram and Comparison Statement:

Initial Class Diagram:



Final Class Diagram:



Key changes to our system:

- Made a UI for the verify page.
- Made a UI for the alert details page
    - Pulls description, type, and location of the alert from the db
- Aggregated list of alerts during app startups for alert display list with a tab for viewing all alerts at once (Event feed).

        - Ordering alerts based on severity: Crime, Warning, Caution
    - Handle location storing while reporting an alert
    - Add pins to map display, representing where reported alerts have occurred.
        - Pins are labeled based on description of the alert
    - Implemented strategy pattern to handle alert similarity
        - Completed cosine similarity and sentiment analysis in caution and warning classes
        - Added method to grab all strings and alert ids from location data for comparison
        - Included similarity algorithm in AlertModal
        - When a new post is created, similarity checker runs similarity score based on strategy algorithm on all alerts of the same type within the location radius
        - When a similarity is found, it fills the verification page with the information it pulls from the db in regards to the already existing (similar) alert. Which then asks the user if they'd rather verify an old alert or create new one.
            - If verified, no new alert is created.
    - Added JUnit tests to verify that our database is being updated correctly and our verification system is functioning as intended (11 tests)

Third Party Code v. Original Code

MapsActivity.java: OnMapReady() function has some code we referenced from https://stackoverflow.com/questions/59576306/how-do-i-restrict-google-maps-api-to-a-specific-area-when-its-latitude-and-longi in regards to how to run and insert effective bounds to the GoogleMaps API.

CautionSimilarity.java: cosine_similarity() function was adapted from https://stackoverflow.com/questions/520241/how-do-i-calculate-the-cosine-similarity-of-two-vectors

Vectorization methods adapted from: https://towardsdatascience.com/understanding-nlp-word-embeddings-text-vectorization-1a23744f7223

DBHandler.java / DBHandlerInterface.java: The SQLite database structure, and corresponding pattern (data object access pattern) design were adapted from these two articles:

https://www.geeksforgeeks.org/how-to-create-and-add-data-to-sqlite-database-in-android/ and https://www.geeksforgeeks.org/data-access-object-pattern/.

Location Permissions request (AlertFormPage.java): The logic to check for location permissions and request them if not given by user was implemented following this YouTube video titled 'Location Permission at Run time":
https://www.youtube.com/watch?v=xwmPXKQ9LJw

Statement on the OOAD process for your overall Semester Project

Our team experienced the Double Diamond model of design throughout working on the semester project. We spent a lot of time finding exactly how to express the problem of notifying a community using a crowd-source based application. In some ways this was positive as we were able to put down many approaches to solving this problem however, a negative impact of this was the amount of time it took to start on finding the right solution for our problem. It took several iterations of finding what were the right patterns and how they should work in the system to pin down the best solution that we had to complete this project. We found that our design process did align with the idea that good architecture design doesn't mean an easy 1:1 transition to software design. What we thought we had planned out well before working in Android studio turned into several workarounds due to the nature of creating software for Android. Not all of our patterns we're as smooth to integrate into the front-end as we had hoped. This came with the negative that we had to scrap some earlier plans for implementing certain patterns but it also positively impacted us by finding that some patterns just weren't necessary as they were almost built into the way one designs Android applications. We used testing as we we're writing the project which worked out in a positive way but could've been more positive if our process included writing test cases at the very beginning of method declaration as we had some hiccups with data type with integrating the project.