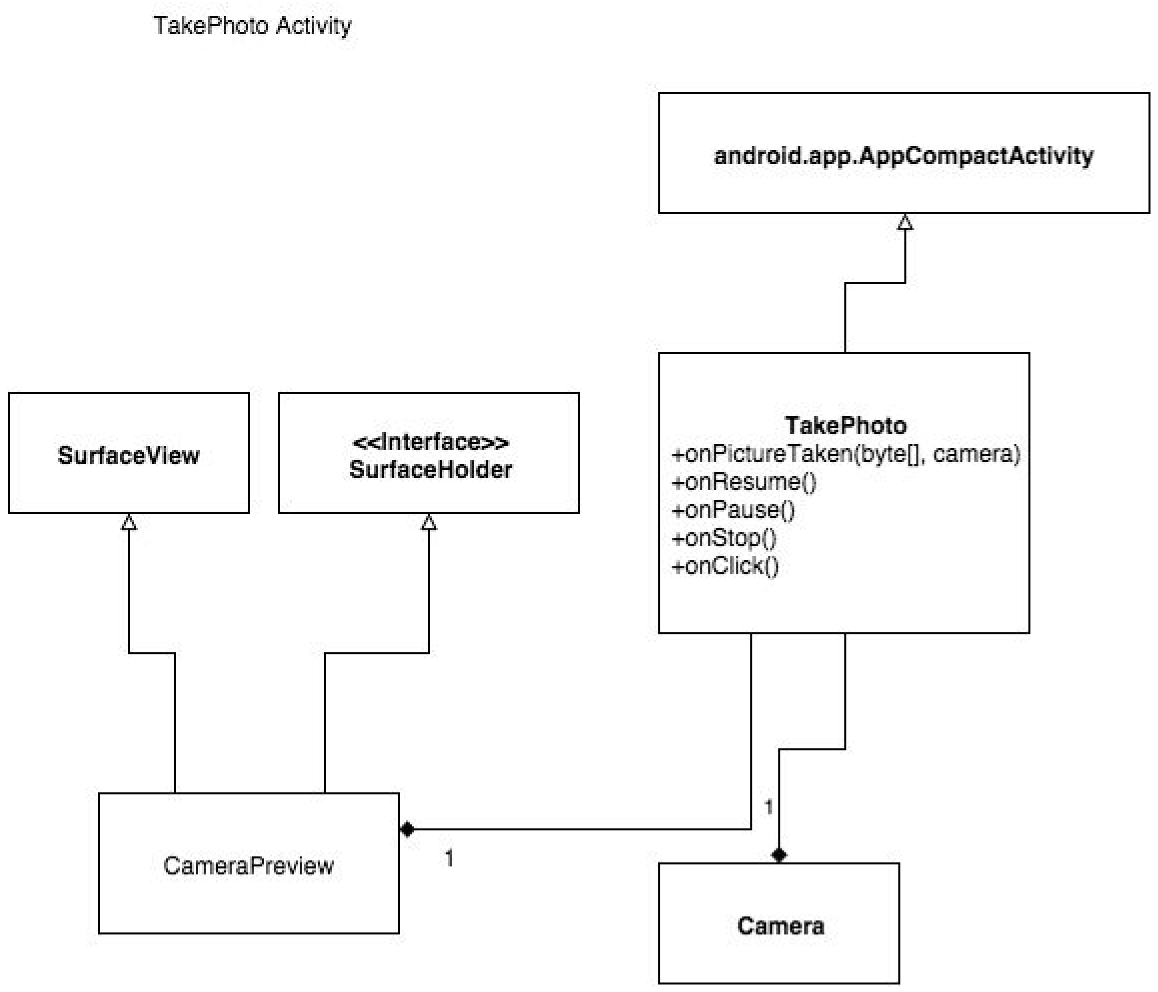
PhotoDroid

William DeRaad, Russ Mehring, Derek Reiersen

Features & Class Diagrams

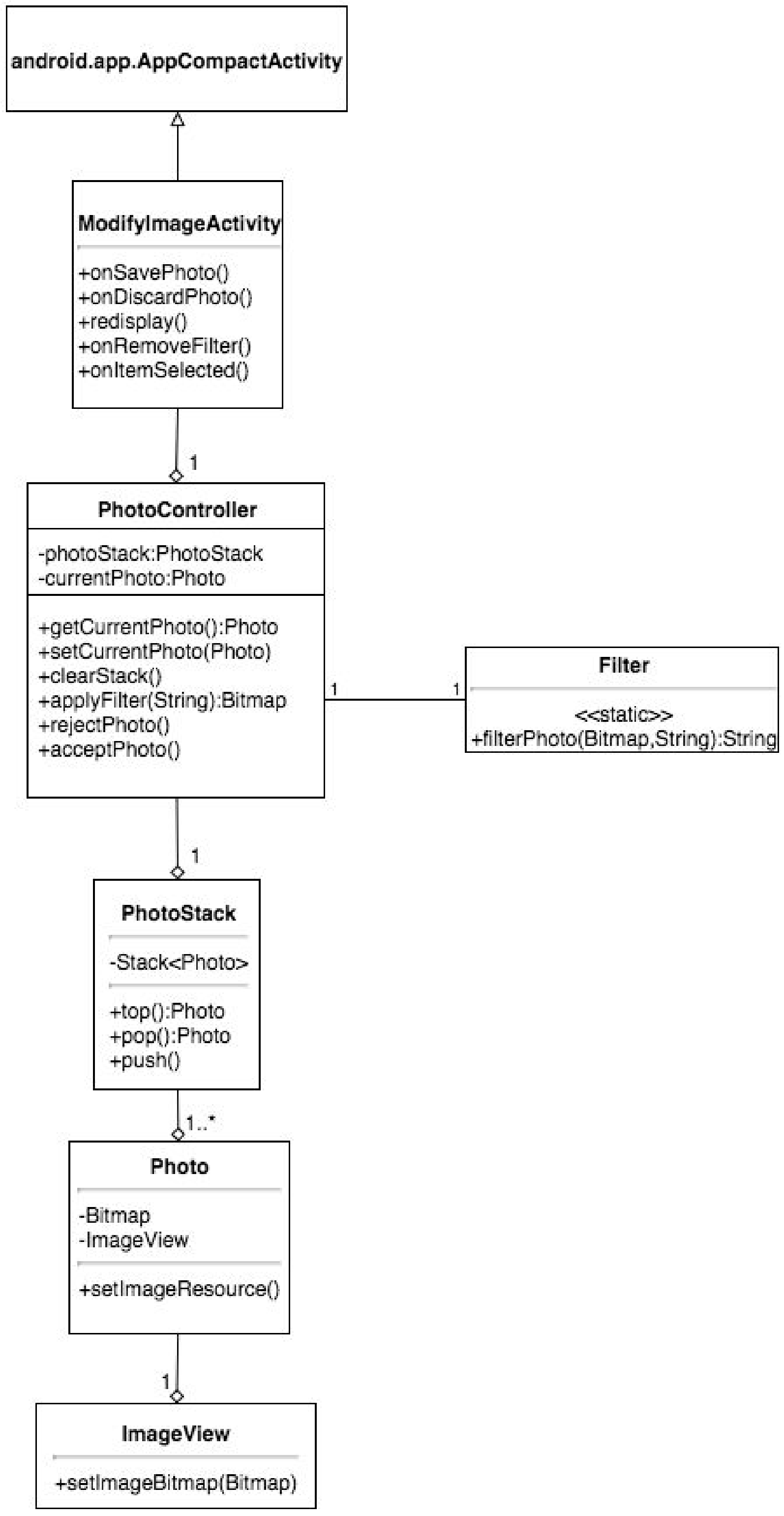
- Planned Features:
    - Take Image
    - ~~Take Pre-Filtered Image~~
    - Load Image from Gallery
    - Modify Image
        - Blur
        - Brighten
        - Contour
        - Dilation
        - Grayscale
        - Erode
        - Laplacian Edge
        - Sharpen
    - Reject Filter
    - Save Image

All of the main features we set out to create were implemented except the ability to pre-filter a photo before it is taken. We altered the planned user interaction required to modify photos slightly, once a filter is applied the user does not need to accept the photo (it is assumed since they did not reject the filtered image), however they still have the option to go backwards and remove filters they have applied.



This class diagram changed throughout development, since it involves all of the system interaction with Android APIs. First, the name was changed from CameraActivity to TakePhoto, as this was a better description of what the activity is doing. Second, in the original class diagram, the surface view was hidden and a Imageview and photo object were used to display the preview image. Since we did not implement the ability to apply a filter to a photo before it is taken, the class was simplified and utilizes the default android way to display a camera preview. The methods in the classes are also slightly different, onResume, onPause, onStop were needed to handle freeing the camera (required by Android). The method onPictureTaken was needed to get the jpeg data once the camera capture in onClick is called.

Manipulate Image Activity Class Diagram



This above class diagram depicts the main functionality of our system, modifying images. The ModifyImage class contains all of the button listener methods that designate the interaction of the user with the system. The PhotoController class is then used to update the model to match the requested interactions by the user. Our model is represented using the PhotoStack, which tracks the changes the user’s make to a particular image. Photos are modified by the user of our static filter library contained within the Filter class. Once the PhotoStack has been updated by our controller the redisplay method is used to update our view with the new state of the system.

Throughout development we needed to change some minor interactions in the above class diagram. Mainly, in our original class diagram (See System Changes) our filter class was accessible through a photo object, instead we chose to implement filtering through a static filter library which the PhotoController interacts with. The rest of our changes were cosmetic to interface with Android and OpenCV (e.g. returning a Bitmap from applyFilter instead of a Photo).

Design Pattern Discussion

The observer design pattern was used to allow the user to press buttons, and the system to respond to these button presses. The Android layout files allow the developer to link buttons in the UI to listener method callbacks in the code. We also use a listener to identify when an item is selected in our filter drop down menu. Without observer, the user would not be able to interact with our system.

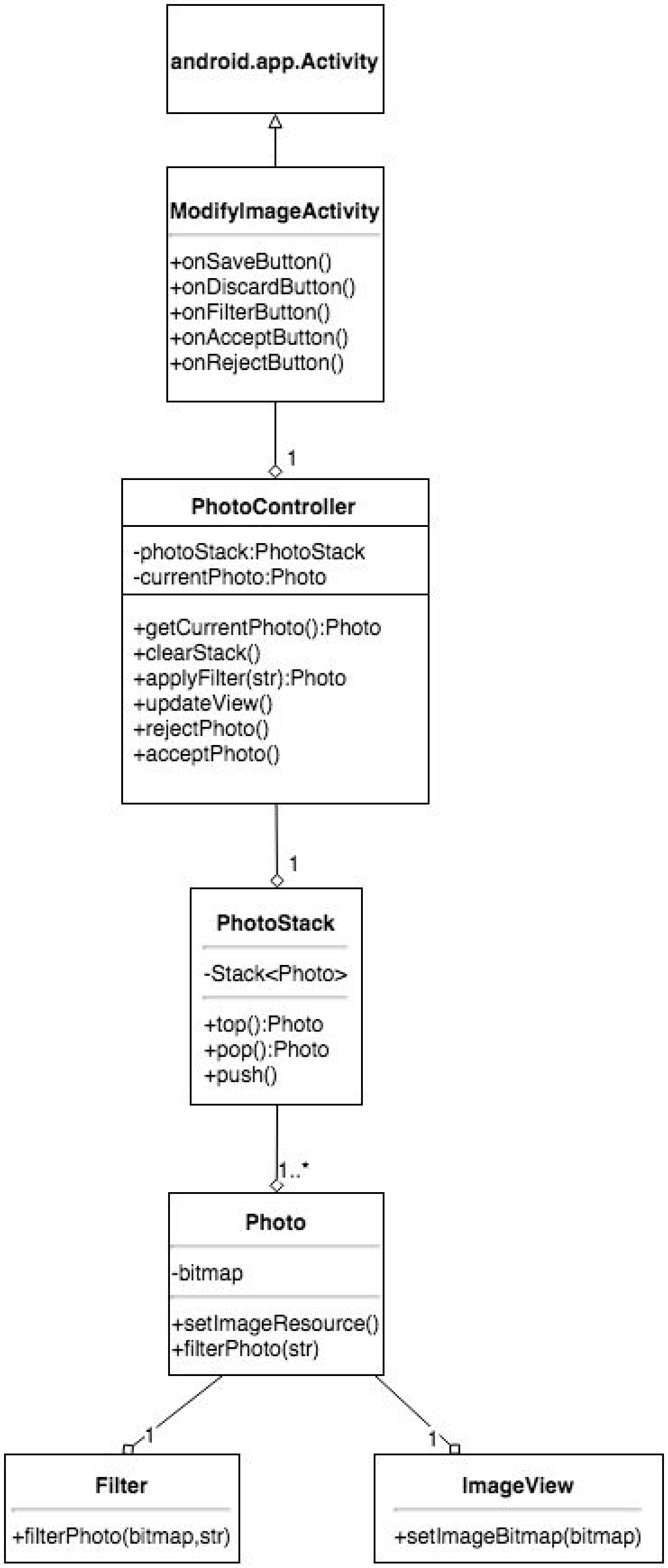
We made use of the memento pattern to implement our reject filter functionality. We save the state of our photo object after each filter storing it in a stack, this allows us to easily undo all the changes a user makes to an image in a fast and efficient manner.

We chose to implement our filter functionality as a static filter library this approach lends itself well to our design, however a more extendable approach for third party developers would have been to use a factory method to choose which filter to use. This would allow third party developers to add new filter classes and then make their new filters available through the factory method.

System Changes

As you can see from the final class diagram (See Features & Class Diagram) and the class diagram below, we were able to implement our system almost identically to original class diagram design. The one major change which was seen earlier is that our filter class was changed to a static filter library access via the PhotoController instead of via the Photo class. The rest of the changes are minor to make development and interaction easier but do not alter the overall structure of our classes.

Manipulate Image Activity Class Diagram



Lessons Learned

- Creating an outline of the system (Project Part 2) made development a lot easier. It was always clear what the next step would be, and it helped break everything into smaller more manageable tasks.
- The class diagram in particular really helped when coding the project, different members could work on different parts of the diagram without stepping on each others toes and still reaching a coherent goal.
- Things can always be subject to change from the original design. What made sense while writing project part 2, made less sense when implementing it. Better options for implementation became apparent when actually coding the problem
- Design patterns are useful, but shouldn't be forced. Not every design pattern applies to a problem, and trying to use a design pattern where it doesn't make sense to use it can complicate things.
- Sometimes what was planned did not work and a different way of achieving the same goal had to be found, realizing this was a key to our success.