MOBILE DEVICES – ASSIGNMENT 3

Mobile Devices – Assignment 3 DevanSuperPhoto

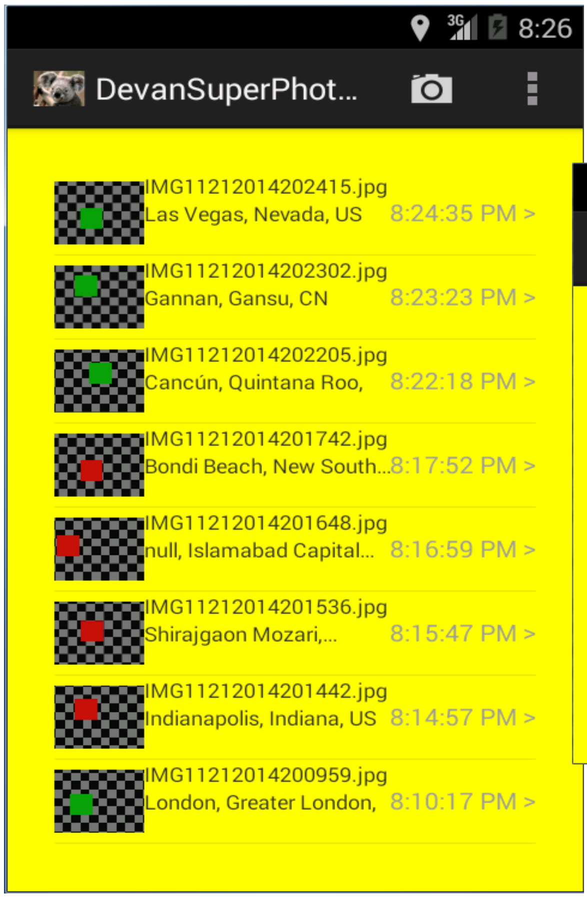
Submitted: November 21, 2014

By: Devan Shah 100428864

Submitted to: Michael Chang

Design Techniques

For assignment 3 I have constructed a photo organizer mobile application in Android using eclipse. Using the interface builder in eclipse for Android I was able to construct the UI design of the Android photo application. My photo application is based off of the default iOS photo application, with the exception of changes in the design to be portable on Android and also reduce meant of the features and the way it works, but the basic concept applies. Before going forward and constructing the design in eclipse I constructed a sample design sketch to see how the photo application would look on a phone. I also added more functionality then required by assignment 3 instructions and also support landscape orientation for most of the views. Also when the orientation is changed it will preserve the data and restore the orientation based of the preserved data. My photo application consists of a simple design with three main transition aspects (views) to move from list of photos to view enlarged photos or add a new photo. My design made use of 2 activity and having the 2 fragments to switch between the views listed above. The list view of photos was implemented as a fragment who's main goal is to make use of a CustomAdapter to populate the list view information in a appropriate manner (ordering by time) and handle onClick. The CustomAdapter was mainly used to display a thumbnail of the photo that was taken. The photo view was implemented as a fragment also, where the main goal was to display the enlarged photo, this fragment also supports deleting a photo. My photo application also supports presentence storage to save the photos that are created so that they can be restored when application is terminated or stopped. This allows for photos to be saved over time until the application is manually uninstalled. Below are some of the snapshots of my design, note that the photos used are simulated.



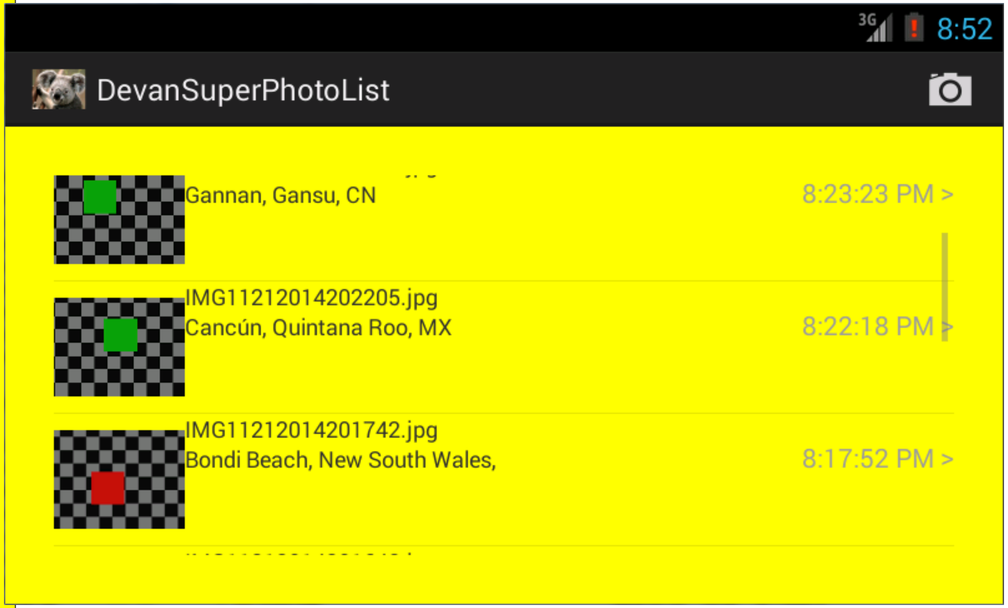
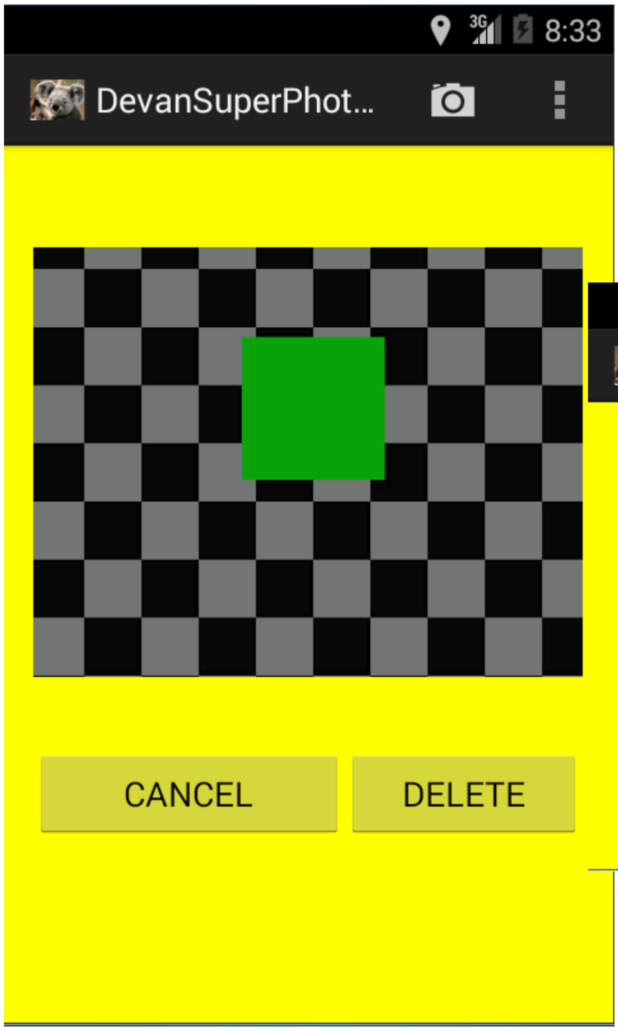


Figure 1 Android Photo UI Design landscape view. Shows list view as scrollable in case all photos don't fit on screen.

Figure 2 Android Photo UI Design portrait view. Shows the list view with multiple photos and also the add button on the right top corner.



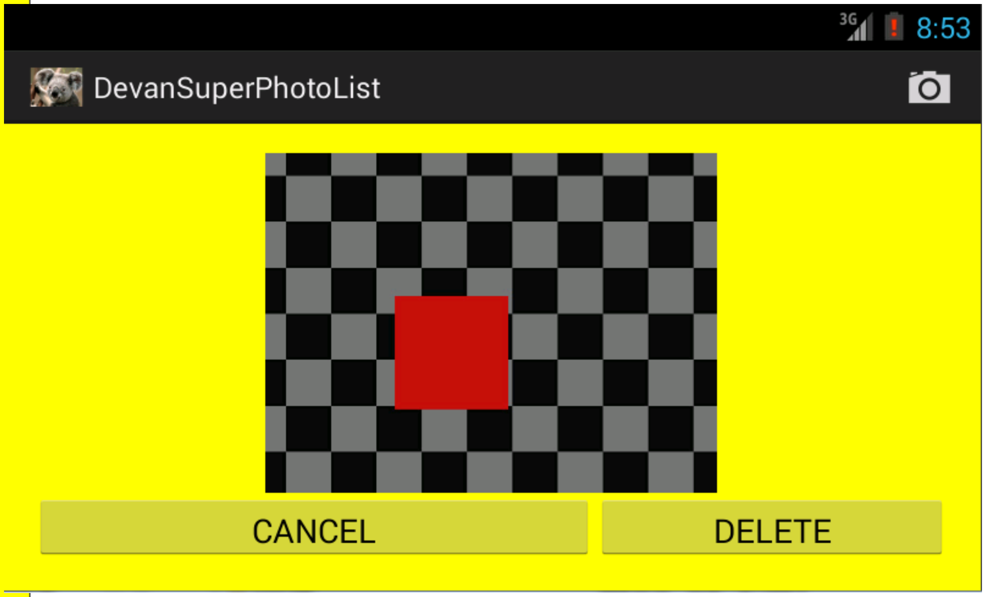
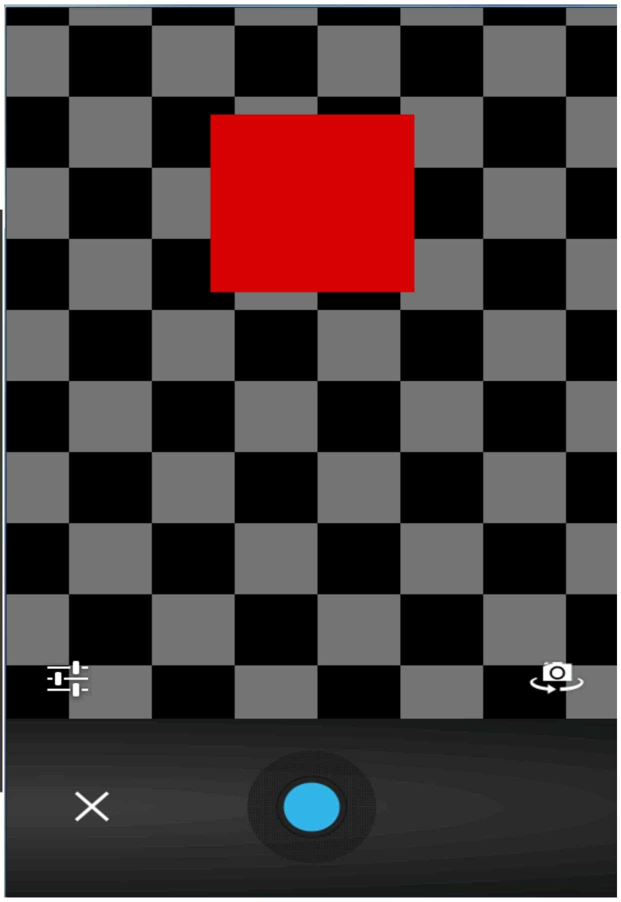


Figure 3 Android Photo UI Design landscape view. Shows the photo view when selected from the list view, this view also supports taking a new photo and adding it to the list.

Figure 4 Android Photo UI Design portrait view. Shows the photo view when selected from the list view, this view also supports taking a new photo and adding it to list.



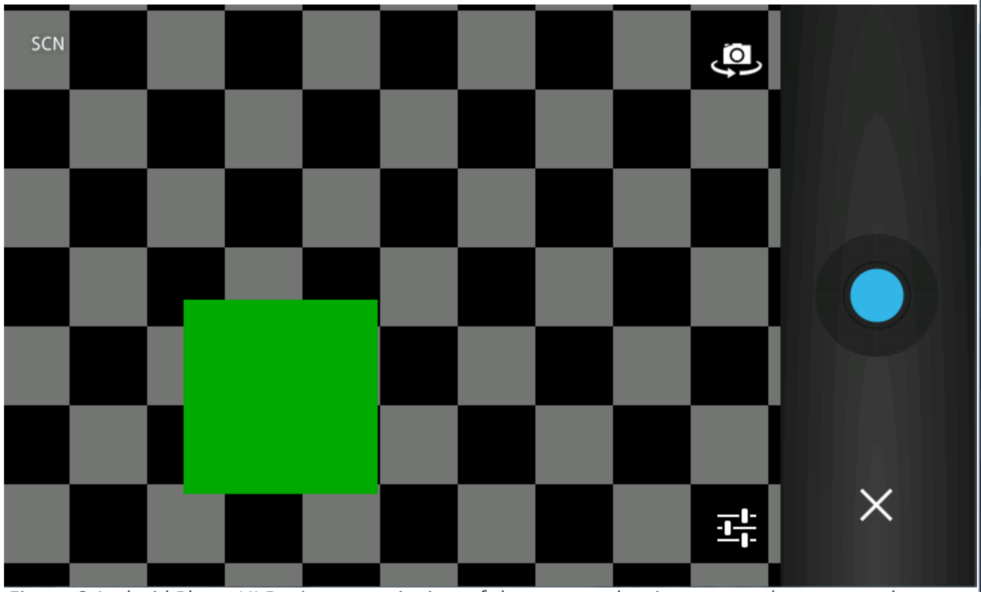


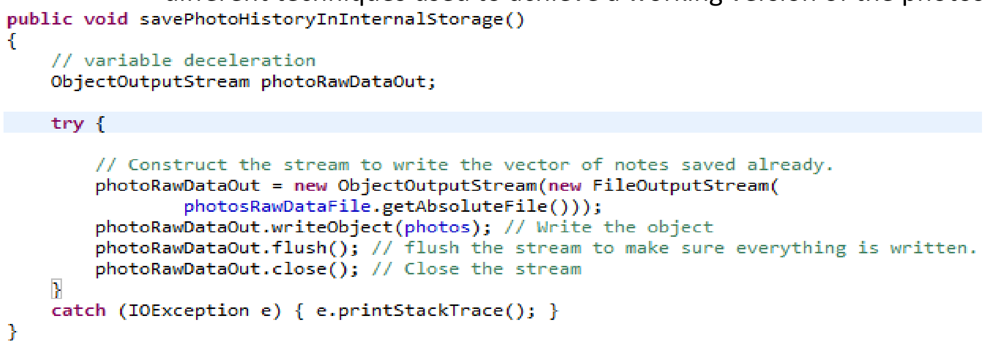
Figure 6 Android Photo UI Design portrait view of the camera that is present when camera button is hit.

Figure 5 Android Photo UI Design landscape view of the camera that is present when camera button is hit.

Implementation Techniques

The Android photo's application was implemented in eclipse using the android application development suite and the use of java. Java played a crucial role in implementing the photo application and having all the operations performed with the use of build in java libraries, default Android objects and google API (google play services). I have implemented multiple functionalities in the photos application which allows it to be robust and perform the most basic operations for saving, deleting or sorting photos. I have implemented couple of error checking mechanisms that are used to make sure errors are prevented down the road.

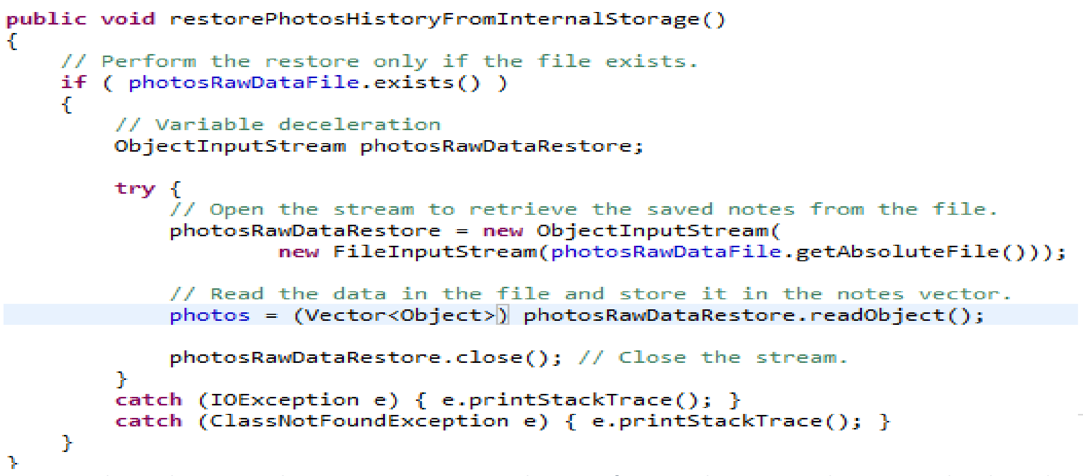
Furthermore on the side of coding techniques I utilized multiple resources on Android development to learn about most efficient ways to implement aspects of the photo application. Couple neat methods that I used to achieve a fully working photo application where the following, Android internal storage, utilized multiple different data structures for storing and displaying information, dialog boxes on Android, different properties of ImageView, list views, text views and fragments. The internal Android storage was used to store the Vector<Object> of PhotosInfo into a file for restore when app crashes or on a new restart. I used multiple different data structure and combinations of data structures, Vector<Object> was used to store the photo/PhotoInfo that were added/deleted, List<PhotoInfo> was used store the photos in a form that could be passed to the list view. Made use of Android dialog boxes to make sure that when the user deletes a photo he/she is prompted when deleting photos. Made use of multiple different xml properties for boxes to cap number of lines, amount of text displayed at once, show there is more text when box length is reached, and different types of input. Furthermore, I made use of fragments for navigation between list of photos and photo view, the fragments are replacing one FrameLayout in the PhotoStart (main Activity). There were multiple different techniques used to achieve a working version of the photos application.



```java
public void savePhotoHistoryInInternalStorage()
{
    // variable deceleration
    ObjectOutputStream photoRawDataOut;

    try {

        // Construct the stream to write the vector of notes saved already.
        photoRawDataOut = new ObjectOutputStream(new FileOutputStream(
                photosRawDataFile.getAbsoluteFile()));
        photoRawDataOut.writeObject(photos); // Write the object
        photoRawDataOut.flush(); // flush the stream to make sure everything is written.
        photoRawDataOut.close(); // Close the stream
    }
    catch (IOException e) { e.printStackTrace(); }
}
```

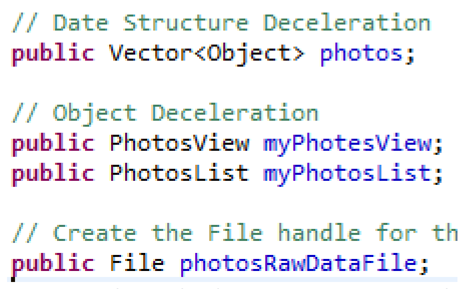
Figure 7 Shows the savePhotoHistoryInInternalStorage function, this function was used to save the photos to android device's internal storage.

Figure 7 and Figure 8 shows snippet of functions that are used to store and read data from Android internal storage for the application. Each application has a dictated internal storage, where application dependent files are stored. I am making use of this functionality to store the photos when new photos info are added/deleted. This makes sure that the photo info for users are always up to date on opening application again after closing.



```java
public void restorePhotosHistoryFromInternalStorage()
{
    // Perform the restore only if the file exists.
    if ( photosRawDataFile.exists() )
    {
        // Variable deceleration
        ObjectInputStream photosRawDataRestore;

        try {
            // Open the stream to retrieve the saved notes from the file.
            photosRawDataRestore = new ObjectInputStream(
                    new FileInputStream(photosRawDataFile.getAbsoluteFile()));

            // Read the data in the file and store it in the notes vector.
            photos = (Vector<Object>) photosRawDataRestore.readObject();

            photosRawDataRestore.close(); // Close the stream.
        }
        catch (IOException e) { e.printStackTrace(); }
        catch (ClassNotFoundException e) { e.printStackTrace(); }
    }
}
```

Figure 6Shows the restorePhotosHistoryFromInternalStorage function that was used to restore the photo data structure from the file stored on internal storage.


```
// Date Structure Deceleration
public Vector<Object> photos;

// Object Deceleration
public PhotosView myPhotesView;
public PhotosList myPhotosList;

// Create the File handle for th
public File photosRawDataFile;
```
Figure 10 Shows the data structure Vector<Object> that holds the PhotoInfo object that users add.

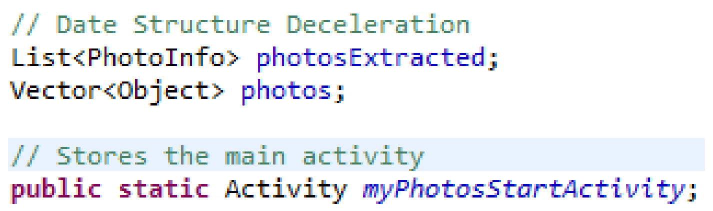
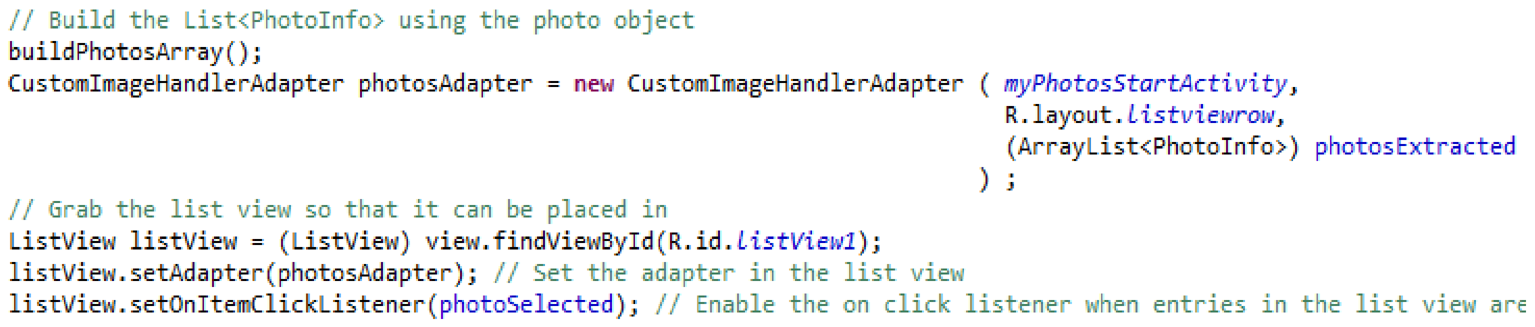

```
// Date Structure Deceleration
List<PhotoInfo> photosExtracted;
Vector<Object> photos;

// Stores the main activity
public static Activity myPhotosStartActivity;
```
Figure 8 Shows the data structure List<PhotoInfo> that is used to store the photos in a form that can be printed to the list view in a proper manner.

Figure 9 and Figure 10 shows the data structure that I used in my photos application, these are the main data structures that are used to store a form of the PhotosInfo objects. These are used to store photos that users add/delete and also used to display the photo in a list view with the use of a CustomAdapter (CustomImageHandlerAdapter) that would use a layout for each of the rows to display thumbnail, filename, geo location and creation date. Figure 10 shows a snippet of how this setup functions to create the list view.


```
// Build the List<PhotoInfo> using the photo object
buildPhotosArray();
CustomImageHandlerAdapter photosAdapter = new CustomImageHandlerAdapter ( myPhotosStartActivity,
                                                                           R.layout.listviewrow,
                                                                           (ArrayList<PhotoInfo>) photosExtracted
                                                                         ) ;

// Grab the list view so that it can be placed in
ListView listView = (ListView) view.findViewById(R.id.listView1);
listView.setAdapter(photosAdapter); // Set the adapter in the list view
listView.setOnItemClickListener(photoSelected); // Enable the on click listener when entries in the list view are
```
Figure 9 Shows how the data structure is used to display the photos in the list view. The CustomAdapter uses the list view row layout, photosExtracted data structure and the main activity. To construct the list view with the bitmap, file name, location and creation date.

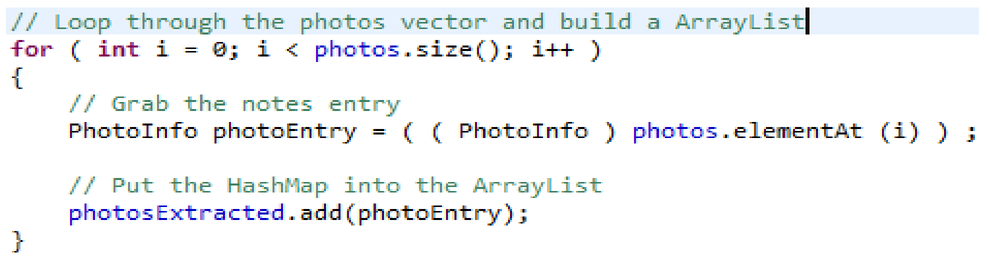
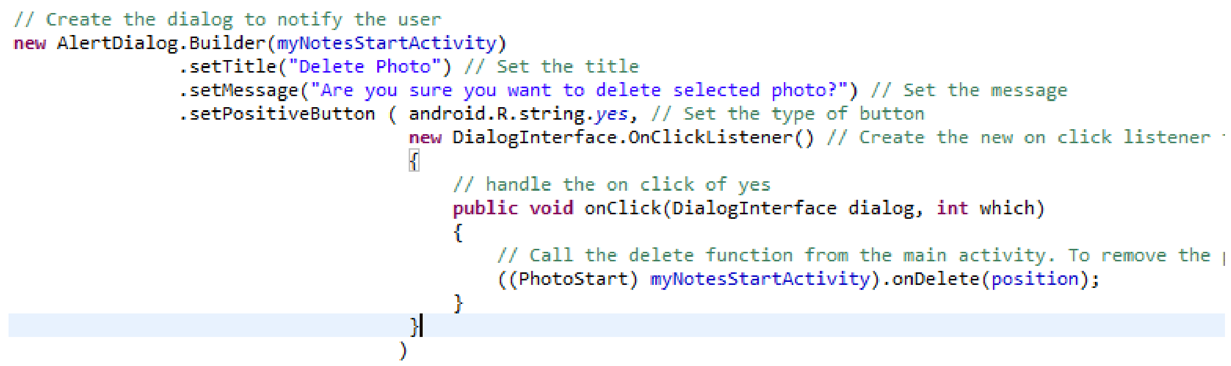
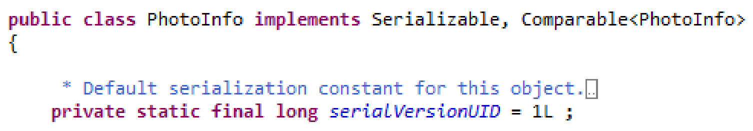

```
// Loop through the photos vector and build a ArrayList
for ( int i = 0; i < photos.size(); i++ )
{
    // Grab the notes entry
    PhotoInfo photoEntry = ( ( PhotoInfo ) photos.elementAt (i) ) ;

    // Put the HashMap into the ArrayList
    photosExtracted.add(photoEntry);
}
```
Figure 11 Shows how the ArrayList<Map<String, String>> is created from the Vector<Object>.

Figure 11 shows the loop that I used to construct the List<PhotoInfo> from the Vector<Object>. The loop runs through the photos and extracts the PhotoInfo objects and puts them into the ArrayList data structure. (This is used by the CustomAdapter to construct the list view.)

Figure 12 shows the snippet of code that I implemented to create the dialog box when the user tries to delete a photo. This is to avoid future failures.

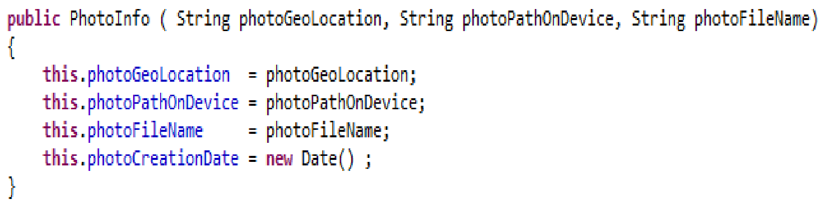

```
// Create the dialog to notify the user
new AlertDialog.Builder(myNotesStartActivity)
                .setTitle("Delete Photo") // Set the title
                .setMessage("Are you sure you want to delete selected photo?") // Set the message
                .setPositiveButton ( android.R.string.yes, // Set the type of button
                                     new DialogInterface.OnClickListener() // Create the new on click listener
                                     {
                                         // handle the on click of yes
                                         public void onClick(DialogInterface dialog, int which)
                                         {
                                             // Call the delete function from the main activity. To remove the
                                             ((PhotoStart) myNotesStartActivity).onDelete(position);
                                         }
                                     }
                                   )
```
Figure 10 Shows how the dialog is constructed when the user want to delete a photo.

I implemented a PhotoInfo object that is used to store all the information for one single photo the user constructs. The PhotoInfo object stores the photos geo location, path on device, file name and created date, I have also implemented multiple functions that are used to retrieve specific aspects of a photo. My PhotoInfo class also implements Serializable and Comparable<PhotoInfo>, the Serializable is used so that the PhotoInfo object can be stored in the file as serialized object and it also implements Comparable<PhotoInfo> so that I can compare the data of PhotoInfo object to determine the order in which to display the Photos in the list view.



```
public class PhotoInfo implements Serializable, Comparable<PhotoInfo>
{

     * Default serialization constant for this object.
    private static final long serialVersionUID = 1L ;
```

Figure 12 Shows the implementation of my PhotoInfo object.

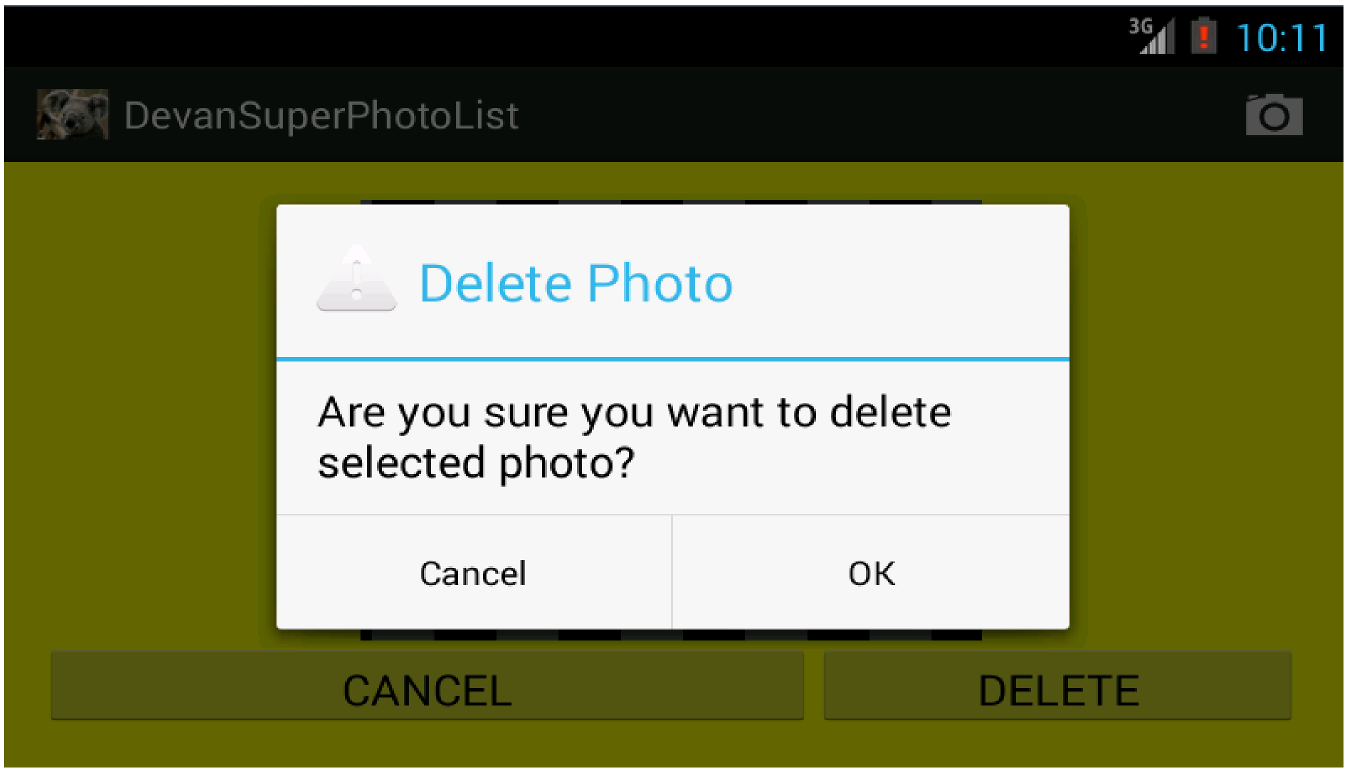


```
public PhotoInfo ( String photoGeoLocation, String photoPathOnDevice, String photoFileName)
{
    this.photoGeoLocation  = photoGeoLocation;
    this.photoPathOnDevice = photoPathOnDevice;
    this.photoFileName     = photoFileName;
    this.photoCreationDate = new Date() ;
}
```

Figure 13 Shows the data that is stored for each of the photos the user creates.

Some of the function that I have implemented in the PhotoInfo class include: getPhotoGeoLocation(), getPhotoDirectory(),getPhotoCreationDate(), getPhotoFileName() and compareTo(PhotoInfo another).

Following is some sample images of features that were implemented aside from design implementations mentioned above.



Are you sure you want to delete selected photo?

Figure 14 Shows the dialog window that opens up when the user tries to delete a photo.

Furthermore, the functionality is pretty self-explanatory when the photo application is being used. The camera button at the top can be used to add new photos, the cancel button on the view photo will return user back to the list view without performing any operations and the delete button on the photo view will delete the photo that was selected.

One major implementation is the use of Google Play Services, LocationManager and Geocoder to retrieve location updates, set location and convert location data into presentable data. (address) To perform this I made use of the LocationUtils, LocationServiceErrorMessage class that were provided by Google as open source android project files. These files were used to handle connections with Google Play Services. I made use of Location Manager as a backup in the case that Google Play Services were not found. The Geocoder was used to convert the longitude and latitude to the closest possible address. No changes were made to the class files LocationUtils, LocationServiceErrorMessage as mentioned by the license agreement from Google. These are mainly error handling messages and global variables for connections actions. Please Note that Google Play Services needs to be available on the machine to make use of this feature or Location Manager will be used instead.

Please refer to the source code for more information on the location accesses using Google Play Services and Location Manager. Main files that handle this are LocationServiceErrorMessage, LocationUtils, PhotoStart, and PhotoLocationManager.