Loopers, Handlers and the HandlerThread-18

You have now downloaded and parsed JSON from the Soo Greyhounds web service, your next task is to download and display images. In this assignment, you will learn how to use Looper, Handler, and HandlerThread to dynamically download and display photos in the Soo Greyhounds app.

1. Create a new repository on GitHub and add your instructor as a collaborator.

2. Open your Soo Greyhounds app in Android Studio. Ensure you have submitted your previous lab. Remove the remote repository that is pointing to your previous lab and add a remote to your new repository. Perform a push to ensure everything is setup properly.

3. The current PhotoHolder in PhotoGalleryFragment simply provides TextViews for the RecyclerView's GridLayoutManager to display. Each TextView displays the caption of a GalleryItem. To display photos, you are going to update PhotoHolder to provide ImageViews instead. Eventually, each ImageView will display a photo downloaded from the mUrl of a GalleryItem. Start by creating a new layout file for your gallery items called list_item_gallery.xml. This layout will consist of a single ImageView setup as follows:

```
                    ImageView
xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/item_image_view"
android:layout_width="match_parent"
android:layout_height="120dp"
android:layout_gravity="center"
android:scaleType="centerCrop"
```

4. Next, update PhotoHolder to hold an ImageView instead of a TextView. Replace bindGalleryItem() with a method to set the ImageView's Drawable.

```
private class PhotoHolder extends RecyclerView.ViewHolder {
~~  private TextView mTitleTextView~~
  private ImageView mItemImageView;
  public PhotoHolder(View itemView) {
    super(itemView);
    ~~mTitleTextView = (TextView) itemView;~~
    mItemImageView = (ImageView) itemView.findViewById(R.id.item_image_view);
  }
~~  public void bindGalleryItem(GalleryItem item) {~~
~~    mTitleTextView.setText(item.toString());~~
~~  }~~
  public void bindDrawable(Drawable drawable) {
    mItemImageView.setImageDrawable(drawable);
  }
}
```

5. Previously the PhotoHolder constructor assumed it would be passed a TextView directly. The new version instead expects a view hierarchy that contains an ImageView with the resource ID R.id.item_image_view. Update PhotoAdapter's onCreateViewHolder(…) to inflate the gallery_item file you created and pass it to PhotoHolder's constructor.

```
public class PhotoGalleryFragment extends Fragment {
  ...
  private class PhotoAdapter extends RecyclerView.Adapter<PhotoHolder> {
      ...
      @Override
      public PhotoHolder onCreateViewHolder(ViewGroup viewGroup, int viewType) {
        ~~TextView textView = new TextView(getActivity());~~
        ~~return new PhotoHolder(textView);~~
        LayoutInflater inflater = LayoutInflater.from(getActivity());
        View view = inflater.inflate(R.layout.list_item_gallery, viewGroup, false);
        return new PhotoHolder(view);
      }
      ...
   }
  ...
}
```

6. Next, you will need a placeholder image for each ImageView to display until you download an image to replace it. Download bill_up_close.png from the LMS and copy it into your res/drawable folder. Update PhotoAdapter's onBindViewHolder(…) to set the placeholder image as the ImageView's Drawable.

```
public class PhotoGalleryFragment extends Fragment {
  ...
  private class PhotoAdapter extends RecyclerView.Adapter<PhotoHolder> {
      ...
      @Override
      public void onBindViewHolder(PhotoHolder photoHolder, int position) {
        GalleryItem galleryItem = mGalleryItems.get(position);
        ~~photoHolder.bindGalleryItem(galleryItem);~~
        Drawable placeholder = getResources().getDrawable(R.drawable.bill_up_close);
        photoHolder.bindDrawable(placeholder);
      }
      ...
   }
  ...
}
```

7. Run Soo Greyhounds, and you should see an array of close-up Bills.

8. Create a new class called ThumbnailDownloader that extends HandlerThread. Then give it a constructor, a stub implementation of a method called queueThumbnail(), and an override of the quit() method that signals when your thread has quit. (Toward the end of the chapter, you will need this bit of information.)

```
public class ThumbnailDownloader<T> extends HandlerThread {
   private static final String TAG = "ThumbnailDownloader";

   private boolean mHasQuit = false;

   public ThumbnailDownloader() {
      super(TAG);
   }

   @Override
   public boolean quit() {
      mHasQuit = true;
      return super.quit();
   }

   public void queueThumbnail(T target, String url) {
      Log.i(TAG, "Got a URL: " + url);
   }
}
```

9. Open PhotoGalleryFragment.java. Give PhotoGalleryFragment a ThumbnailDownloader member variable. In onCreate(…), create the thread and start it. Override onDestroy() to quit the thread.

```
public class PhotoGalleryFragment extends Fragment {

   private static final String TAG = "PhotoGalleryFragment";

   private RecyclerView mPhotoRecyclerView;
   private List<GalleryItem> mItems = new ArrayList<>();
   private ThumbnailDownloader<PhotoHolder> mThumbnailDownloader;
   ...
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setRetainInstance(true);
      new FetchItemsTask().execute();

      mThumbnailDownloader = new ThumbnailDownloader<>();
      mThumbnailDownloader.start();
      mThumbnailDownloader.getLooper();
      Log.i(TAG, "Background thread started");
   }

   @Override
   public View onCreateView(LayoutInflater inflater, ViewGroup container,
         Bundle savedInstanceState) {
      ...
   }

   @Override
   public void onDestroy() {
      super.onDestroy();
      mThumbnailDownloader.quit();
      Log.i(TAG, "Background thread destroyed");
   }
   ...
}
```

10. Within PhotoAdapter.onBindViewHolder(…), call the thread’s queueThumbnail() method and pass in the target PhotoHolder where the image will ultimately be placed and the GalleryItem’s URL to download from.

```
public class PhotoGalleryFragment extends Fragment {
  ...
  private class PhotoAdapter extends RecyclerView.Adapter<PhotoHolder> {
    ...
    @Override
    public void onBindViewHolder(PhotoHolder photoHolder, int position) {
       GalleryItem galleryItem = mGalleryItems.get(position);
       Drawable placeholder = getResources().getDrawable(R.drawable.bill_up_close);
       photoHolder.bindDrawable(placeholder);
       mThumbnailDownloader.queueThumbnail(photoHolder, galleryItem.getUrl());
    }
    ...
  }
    ...
}
```

11. Run Soo Greyhounds and check out Logcat. When you scroll around the RecyclerView, you should see lines in Logcat signaling that ThumbnailDownloader is getting each one of your download requests.

12. Now that you have a HandlerThread up and running, the next step is to create a message with the information passed in to queueThumbnail(), and then put that message on the ThumbnailDownloader’s message queue. First, add the constant and member variables as shown below:

```
public class ThumbnailDownloader<T> extends HandlerThread {
    private static final String TAG = "ThumbnailDownloader";
    private static final int MESSAGE_DOWNLOAD = 0;

    private boolean mHasQuit = false;
    private Handler mRequestHandler;
    private ConcurrentMap<T,String> mRequestMap = new ConcurrentHashMap<>();
    ...
}
```

13. Add code to queueThumbnail(…) to update mRequestMap and to post a new message to the background thread’s message queue.

```
public class ThumbnailDownloader<T> extends HandlerThread {
   private static final String TAG = "ThumbnailDownloader";
   private static final int MESSAGE_DOWNLOAD = 0;

   private boolean mHasQuit = false;
   private Handler mRequestHandler;
   private ConcurrentMap<T,String> mRequestMap = new ConcurrentHashMap<>();

   public ThumbnailDownloader() {
      super(TAG);
   }

   @Override
   public boolean quit() {
      mHasQuit = true;
      return super.quit();
   }

   public void queueThumbnail(T target, String url) {
      Log.i(TAG, "Got a URL: " + url);

      if (url == null) {
         mRequestMap.remove(target);
      } else {
         mRequestMap.put(target, url);
         mRequestHandler.obtainMessage(MESSAGE_DOWNLOAD, target)
               .sendToTarget();
      }
   }
}
```

14. Initialize mRequestHandler and define what that Handler will do when downloaded messages are pulled off the queue and passed to it.

```
public class ThumbnailDownloader<T> extends HandlerThread {
  ...
  @Override
  protected void onLooperPrepared() {
    mRequestHandler = new Handler() {
      @Override
      public void handleMessage(Message msg) {
        if (msg.what == MESSAGE_DOWNLOAD) {
          T target = (T) msg.obj;
          Log.i(TAG, "Got a request for URL: " + mRequestMap.get(target));
          handleRequest(target);
        }
      }
    };
  }

  @Override
  public boolean quit() {
    mHasQuit = true;
    return super.quit();
  }

  public void queueThumbnail(T target, String url) {
    ...
  }

  private void handleRequest(final T target) {
    try {
      final String url = mRequestMap.get(target);

      if (url == null) {
        return;
      }

      byte[] bitmapBytes = new SooGreyhoundsAPI().getUrlBytes(url);
      final Bitmap bitmap = BitmapFactory
            .decodeByteArray(bitmapBytes, 0, bitmapBytes.length);
      Log.i(TAG, "Bitmap created");
    } catch (IOException ioe) {
      Log.e(TAG, "Error downloading image", ioe);
    }
  }
}
```

15. In ThumbnailDownloader.java, add the mResponseHandler variable seen above to hold a Handler passed from the main thread. Then replace the constructor with one that accepts a Handler and sets the variable. Also, add a listener interface that will be used to communicate the responses (downloaded images) with the requester (the main thread).

```java
public class ThumbnailDownloader<T> extends HandlerThread {
    private static final String TAG = "ThumbnailDownloader";
    private static final int MESSAGE_DOWNLOAD = 0;

    private boolean mHasQuit = false;
    private Handler mRequestHandler;
    private ConcurrentMap<T,String> mRequestMap = new ConcurrentHashMap<>();
    private Handler mResponseHandler;
    private ThumbnailDownloadListener<T> mThumbnailDownloadListener;

    public interface ThumbnailDownloadListener<T> {
        void onThumbnailDownloaded(T target, Bitmap thumbnail);
    }

    public void setThumbnailDownloadListener(ThumbnailDownloadListener<T> listener) {
        mThumbnailDownloadListener = listener;
    }

    public ThumbnailDownloader(Handler responseHandler) {
        super(TAG);
        mResponseHandler = responseHandler;
    }
    ...
}
```

16. Modify PhotoGalleryFragment to pass a Handler attached to the main thread to ThumbnailDownloader. Also, set a ThumbnailDownloadListener to handle the downloaded image once it is complete.

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setRetainInstance(true);
  new FetchItemsTask().execute();

  Handler responseHandler = new Handler();
  mThumbnailDownloader = new ThumbnailDownloader<>(responseHandler);
  mThumbnailDownloader.setThumbnailDownloadListener(
    new ThumbnailDownloader.ThumbnailDownloadListener<PhotoHolder>() {
      @Override
      public void onThumbnailDownloaded(PhotoHolder photoHolder,
                                        Bitmap bitmap) {
        Drawable drawable = new BitmapDrawable(getResources(), bitmap);
        photoHolder.bindDrawable(drawable);
      }
    }
  );
  mThumbnailDownloader.start();
  mThumbnailDownloader.getLooper();
  Log.i(TAG, "Background thread started");
}
```

17. In ThumbnailDownloader.handleRequest(), add the following code.

```
public class ThumbnailDownloader<T> extends HandlerThread {


    ...
    private Handler mResponseHandler;
    private ThumbnailDownloadListener<T> mThumbnailDownloadListener;
    ...
    private void handleRequest(final T target) {
      try {
        final String url = mRequestMap.get(target);

        if (url == null) {
          return;
        }

        byte[] bitmapBytes = new SooGreyhoundsAPI().getUrlBytes(url);
        final Bitmap bitmap = BitmapFactory
            .decodeByteArray(bitmapBytes, 0, bitmapBytes.length);
        Log.i(TAG, "Bitmap created");

        mResponseHandler.post(new Runnable() {
          public void run() {
            if (mRequestMap.get(target) != url ||
                mHasQuit) {
              return;
            }

            mRequestMap.remove(target);
            mThumbnailDownloadListener.onThumbnailDownloaded(target, bitmap);
          }
        });

      } catch (IOException ioe) {
        Log.e(TAG, "Error downloading image", ioe);
      }
    }
  }
```

18. Before running Soo Greyhounds and seeing your hard-won images, there is one last danger you need to account for. If the user rotates the screen, ThumbnailDownloader may be hanging on to invalid PhotoHolders. Bad things will happen if the corresponding ImageViews get pressed. Write a clearQueue() method to clean all the requests out of your queue.

```
public class ThumbnailDownloader<T> extends HandlerThread {
  ...
  public void queueThumbnail(T target, String url) {
    ...
  }

  public void clearQueue() {
    mRequestHandler.removeMessages(MESSAGE_DOWNLOAD);
    mRequestMap.clear();
  }

  private void handleRequest(final T target) {
    ...
  }
}
```

19. Then clean out your downloader in PhotoGalleryFragment when your view is destroyed.

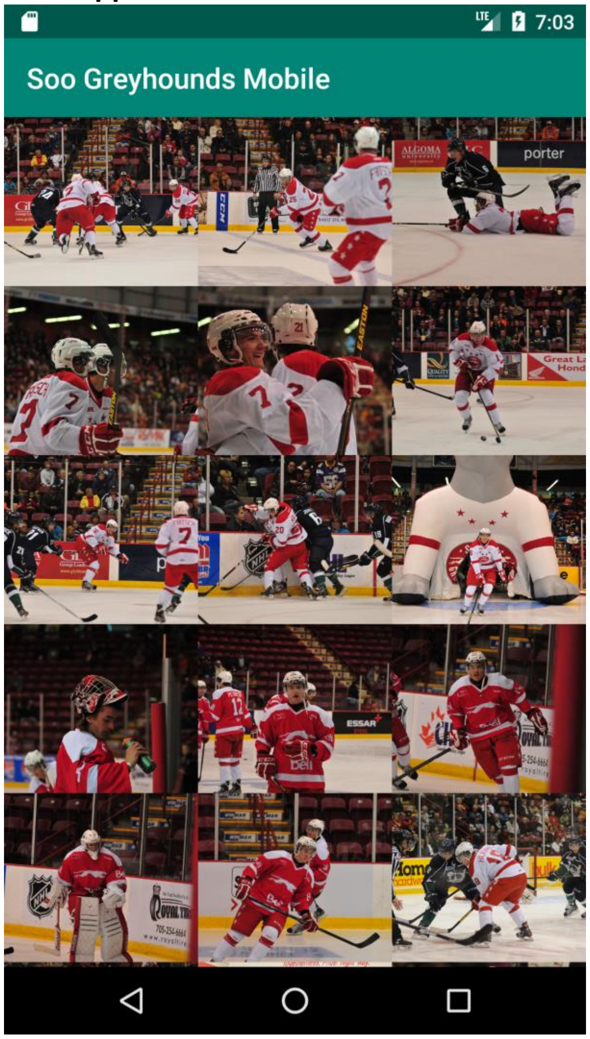
```
public class PhotoGalleryFragment extends Fragment {
 ...
 @Override
 public View onCreateView(LayoutInflater inflater, ViewGroup container,
                          Bundle savedInstanceState) {
    ...
 }

 @Override
 public void onDestroyView() {
    super.onDestroyView();
    mThumbnailDownloader.clearQueue();
 }

 @Override
 public void onDestroy() {
    ...
 }
 ...
}
```

20. Run Soo Greyhounds. Scroll around to see images dynamically loading.

Final App


7:03
Soo Greyhounds Mobile