CMSC436: Fall 2013 – Week 6 Lab

Objectives:

Familiarize yourself with User Notifications and the BroadcastReceiver class. Create two applications using notifications and BroadcastReceiver.

Once you've completed this lab you should have a better understanding of User Notifications and the BroadcastReceiver classes. You should know how to create and display different types of User Notifications to inform a user of the application's actions. You should also know how to broadcast Intents in different ways.

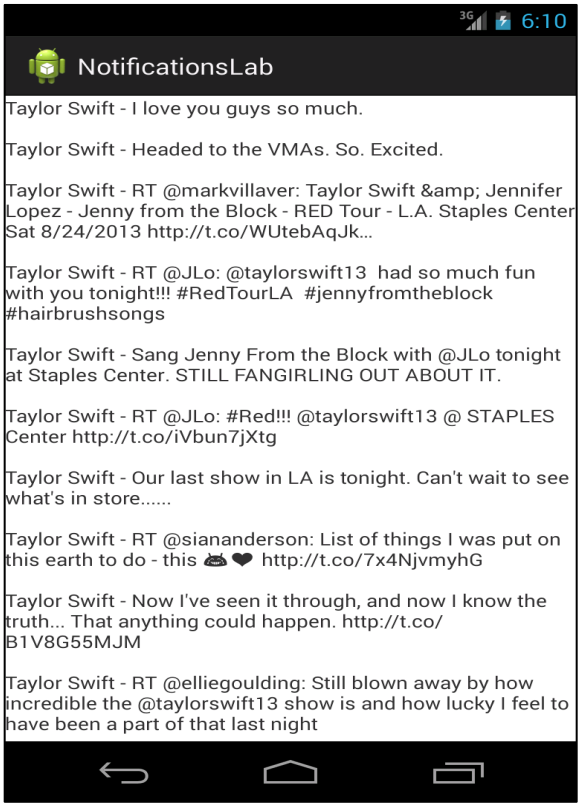
Overview:

This lab has two parts. One that focuses primarily on User Notifications (but uses the BroadcastReceiver class, and one that focuses primarily on the BroadcastReceiver class.

Part 1: User Notifications

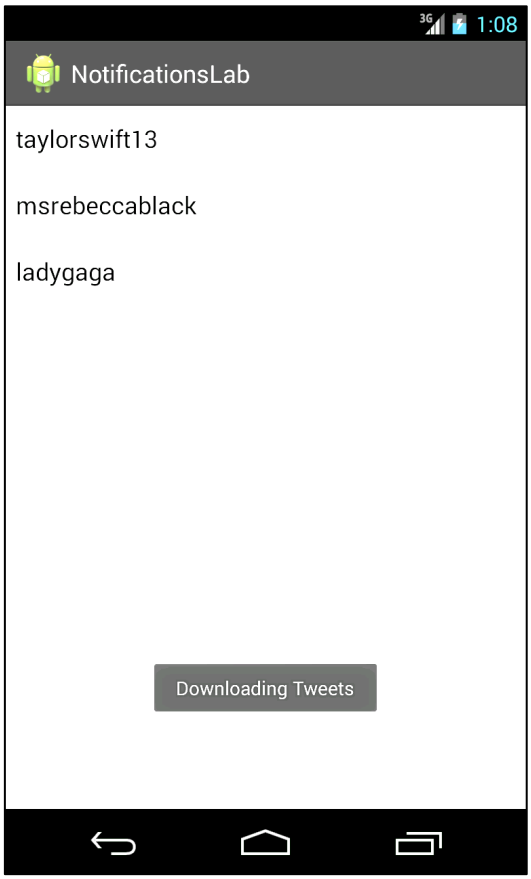
Exercise A: User Notifications

This lab is a modified version of the previous lab using Twitter data. Thus, its interfaces and code are quite similar to those of the previous lab.

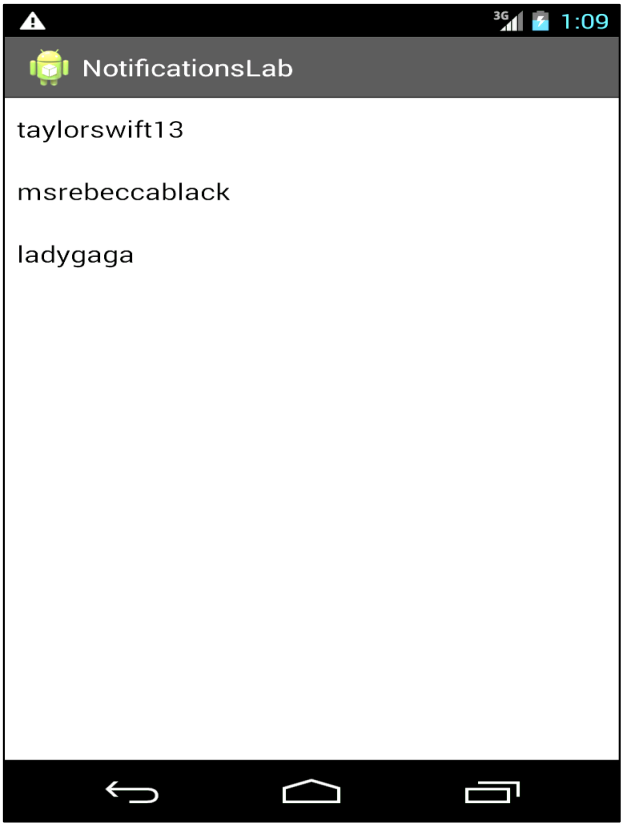


When the application runs it downloads new Twitter data. For now, this "downloading" is only simulated. We'll add downloading actual data from the network in a later lesson. Because this downloading might take some time, this lab will add User Notifications to keep the user apprised of the application's downloading progress.

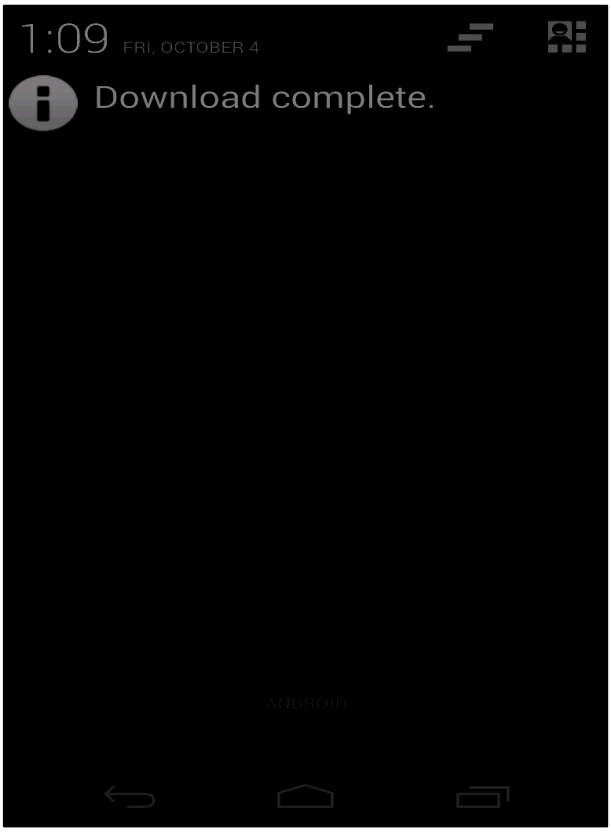
Specifically, this version of the application uses a Service, which we'll provide, to download Twitter data. Around the time when the Service starts the download, the application should create a Toast message to alert users that the download is starting. An example is shown below:



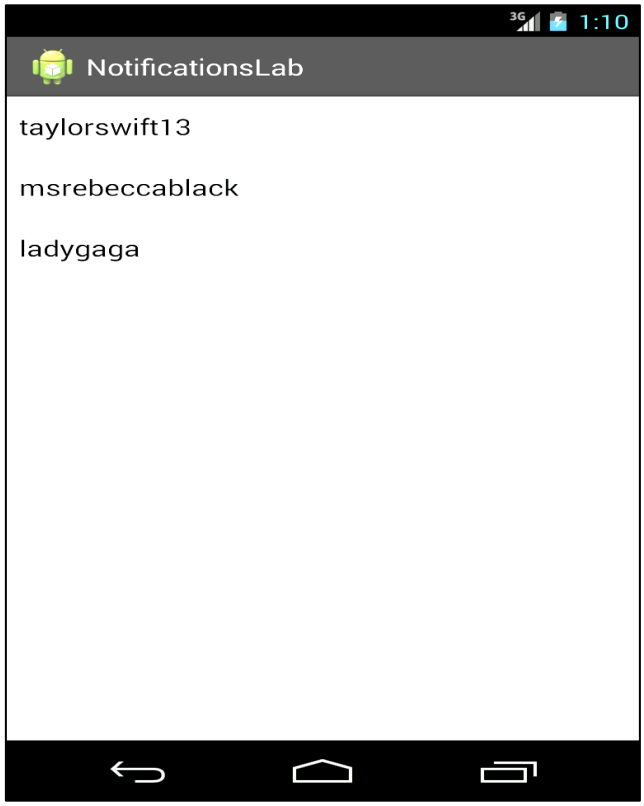
Because download time can take vary, this application will notify the user when the download is complete. This notification will come as a Notification Area Notification, similar to what's shown below:



If the user pulls down on the Notification drawer he or she will see some indication of whether or not the download was successful.



If the user clicks on the Notification Drawer View, the application should re-open to display the main view



The User Notification behavior should appear and function in the same manner when the application is running in tablet mode.

Outside of the Notification behaviors, this application should generally work the same way as the previous lab. One additional difference is that if the use selects an item in the friends list before the Twitter data is available, no action is taken.

Exercise B: SendBroadcast

In this exercise, you will add a BroadcastReceiver to the Main Activity. You will dynamically register it to receive a Broadcast from the Service indicating that it has finished downloading the Twitter data. When the BroadcastReceiver receives this Intent, it will enable the FriendsFeed Listview, so that when the user selects an item in the ListView, the corresponding Tweets can be displayed. Note that this behavior should only occur if the application is running when the Intent is broadcast.

Exercise C: SendOrderedBroadcast

In exercise when the Service finishes downloading the Twitter data, it sends a User Notification and it broadcasts an Intent. If the application is running when the Intent is broadcast, then the User Notification is redundant and leaves an icon in the Notification Area that the user will need to dismiss. In this exercise you will use the SendOrderedBroadcast() method to broadcast an Intent, passing its own BroadcastReceiver to receive the result of the broadcast. If the Intent is received by the

BroadcastReceiver you created in exercise B, then it will should set some result so that the Service knows that application's alive. If the application's still alive then the Service should not send the User Notification.

Implementation Notes:

1. Download the application skeleton files from the Lectures & Labs page and import it into your IDE.
2. We are providing you with the source code for the download service which is implemented in DownloadService.java
3. For exercise A, implement the following classes and methods
   In MainActivity.java.
   a. Modify the private void ensureData() method to create a Toast message saying "Downloading Tweets"
   In DownloadService.java.
   b. Modify the private void notify(boolean success) method to create a notification in the notifications area to alert the user that the downloading has completed.
4. For Exercise B, implement the following classes and methods
   In MainActivity.java.
   a. Modify the private void ensureData() method to add a BroadcastReceiver that enables the FriendsFeed Listview once the download service finishes downloading.
   In DownloadService.java.
   b. Modify the private void notify(boolean success) method to add a sendBroadcast() call to the BroadcastReceiver above. You should use sendBroadcast method from LocalBroadcastManager. More information about LocalBroadcastManager can be found at:
   http://developer.android.com/reference/android/support/v4/content/LocalBroadcastManager.html
5. For Exercise C, implement the following classes and methods
   In MainActivity.java.
   a. Modify the private void ensureData() method to make the BroadcastReceiver in Exercise B returns a result to inform the Service that the application's still alive. If so, the Service should not send the user notification.
   In DownloadService.java.
   b. Modify private void notify(boolean success) method to replace the sendBroadcast in Exercise B by a sendOrderedBroadcast(). Make additional changes as needed to return a result to the Service and to take action based on the result. More information about sendOrderedBroadcast can be found at:
   http://developer.android.com/reference/android/content/Context.html

Deliverables:

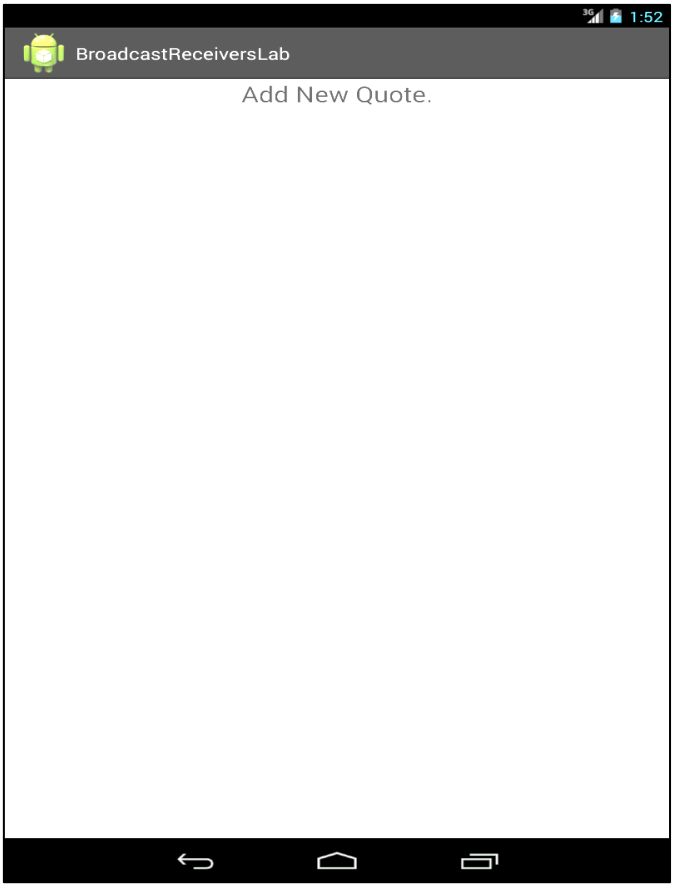
Submit your lab via the submit server. Your submission should include a zip file containing the three projects that implement Exercises A, B and C.

Part2: More BroadcastRecievers

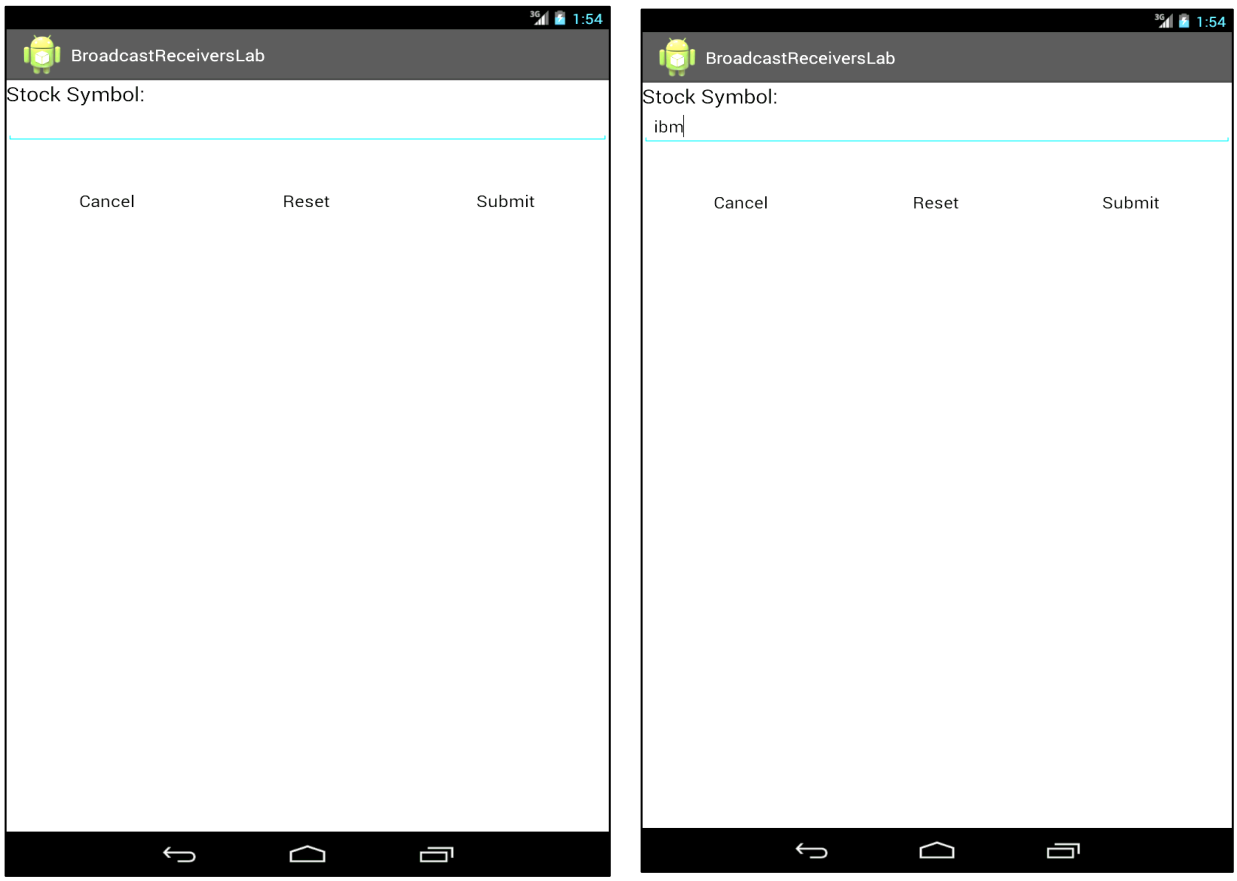
This lab is a simple Stock Quote Viewer. Since we haven't done networking yet, all the stock prices are initially generated randomly (an int between 100-200). These prices will increase by 1 every time an update is requested.

Since battery management is important in mobile applications, we'll be implementing a Broadcast Receiver that adjusts its updating behavior based on the status of the device's battery.

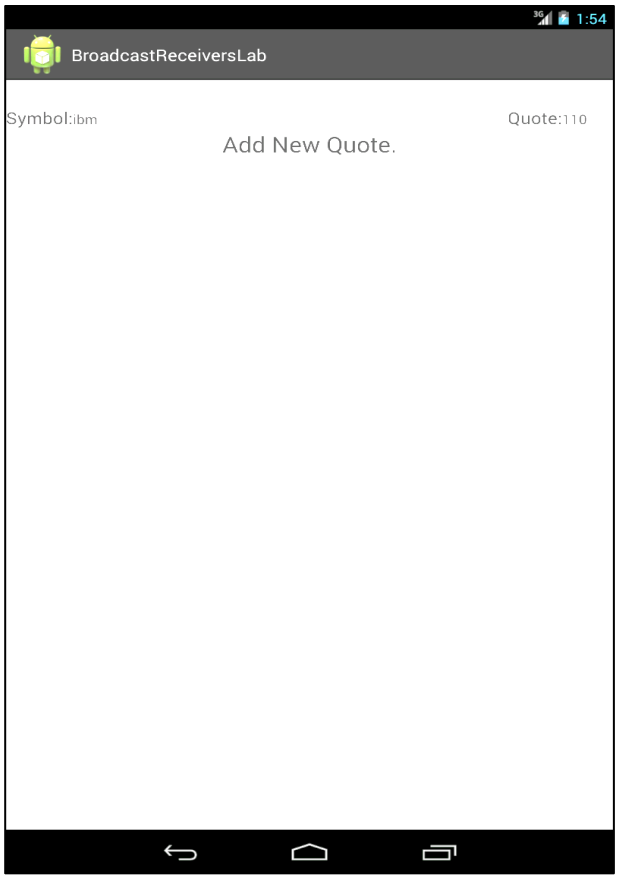
The application's initial interface is shown below. Like the ToDo manager application, this one presents a ListView, with a special footer for adding new information.



When the app starts for the very first time ever, there will be no quotes. When the user clicks the Add New Quote view, another interface will be presented, allowing the user to add a new stock quote Symbols that application will then display and update. That interface will look something like this:



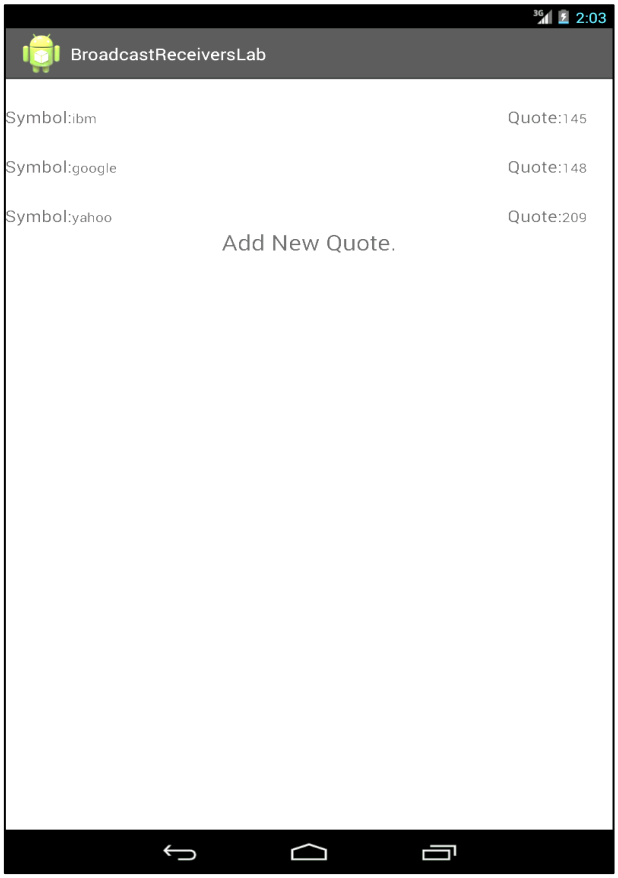
The Cancel, Reset and Submit buttons have their normal interpretation. Suppose the user entered "IBM" hits the Submit button and gets a stock quote of 110. In that case the application's display should look something like this:



After two “updates” the quote should be 112.

You will implement a BroadcastReceiver to filter intents from the system so the application is alerted when the battery enters a Low/Critical state and when it returns to an OK state. When battery enters the Low state, the application should disable updates. When the battery returns to the OK state, the application should run updates every 15 seconds. Note that when the application starts up for the first time, updating should be enabled only if the current battery level is >20%.

The final product should look something like this:



In order to test the application, you will need to alter the battery's state. This will be easiest to do if you run in the emulator. You can change the emulator state by starting a telnet session with the emulator. You can provoke a Low battery state, by first turning on ac power to the emulator (power ac off) and then by setting the battery level to a low value (power capacity 5). Reverse the operations to exit the low battery state. See http://developer.android.com/guide/developing/tools/emulator.html for more information on the emulator.

Implementation Notes:

1. Download the application skeleton files from the Lectures & Labs web page and import them into your IDE. The skeleton provides most of the app's implementation except for the BroadcastReceiver.

2. In StockQuoteActivity.java, go to TODO item and implement, mBatteryReceiver, the BroadcastReceiver which will process different cases of the battery state (Hint: You only need to process two cases corresponding two actions: Intent.ACTION_BATTERY_LOW and Intent.ACTION_BATTERY_OKAY).

3. In StockQuoteActivity.java, go to TODO item and implement the logic that determines whether or not to turns on the updating.

Deliverables: - your source code project