NotifyMe Git
https://github.com/chrisayoola2/NotifyMe.git

Sensor Git
https://github.com/chrisayoola2/SensorSurvey.git

Notification Scheduler Git
https://github.com/chrisayoola2/NotificationScheduler.git

Sensor Lab

The sensor manager is a system service that lets you access the device sensors.

The getDefaultSensor() method is used to query the sensor manager for sensors of a given type.

When sensor data changes, the Android sensor framework generates an event (a SensorEvent) for that new data. Your app can register listeners for these events, then handle the new sensor data in an onSensorChanged() callback. All of these tasks are part of the SensorEventListener interface.

SensorEventListener interface includes two callback methods that enable your app to handle sensor events:

- onSensorChanged(): Called when new sensor data is available. You will use this callback most often to handle new sensor data in your app.
- onAccuracyChanged(): Called if the sensor's accuracy changes, so your app can react to that change. Most sensors, including the light and proximity sensors, do not report accuracy changes. In this app, you leave onAccuracyChanged() empty

Use the onStart() and onStop() methods to register and unregister your sensor listeners.”

NotifyMeLab

For each notification channel, your app sets *behavior* for the channel, and the behavior is applied to all the notifications in the channel. For example, your app might set the notifications in a channel to play a sound, blink a light, or vibrate.

channel ID. Every notification channel must be associated with an ID that is unique within your package. You use this channel ID later, to post your notifications.

. The PendingIntent allows the Android notification system to perform the assigned action on behalf of your code.
Notifications are created using the NotificationCompat.Builderclass

A *notification* is a message that you can display to the user outside of your app's normal UI:

- Notifications provide a way for your app to interact with the user even when the app is not running.
- When Android issues a notification, the notification appears first as an icon in the notification area of the device.
- To specify the UI and actions for a notification, use NotificationCompat.Builder.
- To create a notification, use NotificationCompat.Builder.build().
- To issue a notification, use NotificationManager.notify() to pass the notification object to the Android runtime system.
- To make it possible to update or cancel a notification, associate a notification ID with the notification.
- Notifications can have several components, including a small icon (setSmallIcon(), required); a title (setContentTitle()); and detailed text (setContentText()).
- Notifications can also include pending intents, expanded styles, priorities, etc. For more details, seeNotificationCompat.Builder.

Which API do you use to add an action button to a notification?

- NotificationManager.addAction()

JobScheduler

https://codelabs.developers.google.com/codelabs/android-training-job-scheduler/index.html?index=..%2F..android-training#2

To use JobScheduler, you need to use JobService and JobInfo:

- A JobInfo object contains the set of conditions that trigger a job to run.
- A JobService is the implementation of the job that runs under the conditions set in the JobInfo object.

onStartJob() Returns a boolean indicating whether the job needs to continue on a separate thread. If true, the work is offloaded to a different thread, and your app must call jobFinished()

If false, the system knows that the job is completed by the end of onStartJob(), and the system calls jobFinished() on your behalf.

onStopJob() callback:If the conditions described in the JobInfo are no longer met, the job must be stopped, and the system calls onStopJob().

The onStopJob() callback returns a boolean that determines what to do if the job is not finished. If the return value is true, the job is rescheduled; otherwise, the job is dropped.

- obScheduler provides a flexible framework to intelligently accomplish background services.
- JobScheduler is only available on devices running API 21 and higher.
- To use the JobScheduler, you need two parts: JobServiceand JobInfo.
- JobInfo is a set of conditions that trigger the job to run.
- JobService implements the job to run under the conditions specified by JobInfo.
- You only have to implement the onStartJob() and onStopJob() callback methods, which you do in your JobService.
- The implementation of your job occurs, or is started, in onStartJob().
- The onStartJob() method returns a boolean value that indicates whether the service needs to process the work in a separate thread.
- If onStartJob() returns true, you must explicitly call jobFinished(). If onStartJob() returns false, the runtime calls jobFinished() on your behalf.
- JobService is processed on the main thread, so you should avoid lengthy calculations or I/O.
- JobScheduler is the manager class responsible for scheduling the task. JobScheduler batches tasks to maximize the efficiency of system resources, which means that you do not have exact control of when tasks are executed.