Compass Design using Sensors Programmers Guide

## Introduction

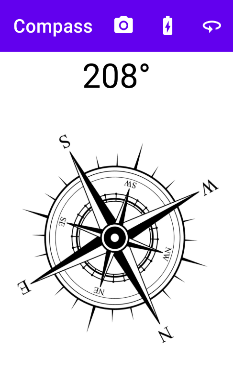
This document is designed to help other programmers understand how the software works. It goes over the functionality, UI design and the architecture. The application is complied for SDK 29, min SDK 16 and targeted SDK 29. The Application is set out so that the first screen is the CompassActivity, containing a compass animation and a textview displaying the orientation (bearing) in 360 degrees. From here the user can use the toolbar icons to select different functionality:

* Camera: If pressed, this allows the user to take a picture and saves it to either the SD card (if available) or the local gallery under “EMFtrackingDevice”.
* Battery: If pressed, this changes the activity to the BatteryActivity unless already in it.
* Compass: If pressed, this changes the activity to the CompassActivity unless already in it.

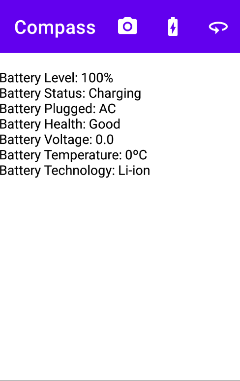
The use of the toolbar as the interaction for the user allows for easy changes to any implementation whilst keeping the core functionality available should anyone choose to add extra functionality such as buttons etc.

### 

* Battery information presented inside the textview.
* Animation that points to North, even when the device is rotated.



CompassActivity Interface



BatteryActivity Interface

## Sensors implementation

There is only one sensor explicitly used which is the **rotation vector** which “Measures the orientation of a device by providing the three elements of the device's rotation vector.” (Android, 2019). This is chosen because it quickly and easily allows for the orientation of the device. It needs the accelerometer and magnetometer to function but will also use the gyroscope to increase the accuracy.

Everything to do with getting the bearing is done inside the Orientation class. This class is a listener to the sensor class such that when the rotation vector’s accuracy or value changes, this class is notified. It also implements an observer design pattern by allowing the activities to listen in on any orientation changes (Tutorialspoint, 2020). Thus when a sensor notification is given, the class updates the new orientation and alerts any activity that is attached as a listener.

## Sensor data processing

The only sensor data processing consists of getting the rotation matrix from the rotation vector and using only the first value returned, which is in radians. This is then converted into degrees which is between [-180,180] and after this is done, listeners are notified of the new bearing. This whole process is done inside the method **calculateOrientation().** There is the method, **convertTo360Degrees(),** included which will convert the angle to be between [0,360]. This provides strong coupling within the architecture, leaving the Orientation class to deal with everting compass related.

## Principal methods & listeners

As already mentioned, the Orientation is a listener for sensor data and the activities are listeners for orientation data.

There is also the listener for battery information contained within the **BatteryActivity** activity. This is called using the **registerReciever()** passing in the Intent for battery information. Thus when the battery information changes, the **batteryReciever()** method is executed which then executes the **Battery** class’s **setBatteryValues()** method. This use of this battery class allows for low coupling and strong cohesion due to this class dealing with the battery results and the activity is less cluttered.

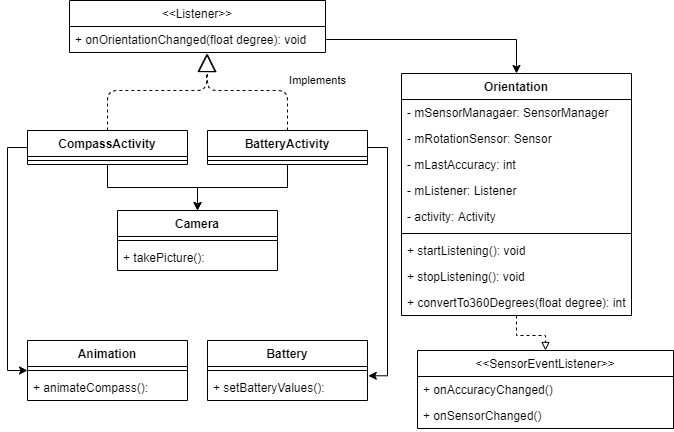
There is an Animation class which is included that deals with all of the animation for the compass to point in the correct direction when the phone moves. This allows better cohesion in the CompassActivity. The main method to call here is the **animateCompass()**, which deals with the animation.

There is a Camera class which deals with the taking a picture by determining the Uri fileUri object needed to save the image to storage. The main method called is the **takePicture()**, which is called whenever the user wants to take a picture (presses the camera icon in the toolbar). In order to use the camera, the user needs to grant permissions: Camera, read storage and write storage. These are queried when the user wants to take a picture through the **checkCameraPermissions().** If the user does not grant permissions, then the camera activity is not started.

## Coding style

The coding practices adhere to the javadoc standard where parameters, returns and general statements for methods are provided (Oracle, 2020). The constants are all upper case, the global variables are defined as first letter lowercase, next word first letter uppercase. Modularity has already been described with the extra class created.

## Extra features

The use of the toolbar for easy implementation and simple extension is considered the extra functionality as already described in section 1.2.

UML class diagram

# **Bibliography**

Android. (2019, 12 27). *Sensors Overview.* Retrieved from Android: https://developer.android.com/guide/topics/sensors/sensors\_overview

Oracle. (2020, 02 02). *How to Write Doc Comments for the Javadoc Tool*. Retrieved from Oracle: https://www.oracle.com/uk/technical-resources/articles/java/javadoc-tool.html

Tutorialspoint. (2020, 02 02). *Design Patterns - Observer Pattern*. Retrieved from tutorialspoint: https://www.tutorialspoint.com/design\_pattern/observer\_pattern.htm