Programmer's guide

by Galiț Ilie: s1628465

Introduction

This guide is aimed at providing a technical overview of the Compass Application developed for the EWireless Course. The Compass App is required to show an image of a compass that reflects the current device orientation, the value of the orientation along with its direction, take a picture and save it with its name consisting of the orientation and direction. The app is also required to display the device's battery information.

Structure, Modularity & Coding Style

The application consists of 3 activities: Compass Activity, Gallery Activity and Battery Activity. *Figure 1 and 2* show the application workflow and how the modules are structured and interact. The reasons for separating the application features into 3 different activities are:

- *Good Software design practices*: each activity is responsible for a separate unique feature that the App provides, thus splitting them in separate modules contributes to having high cohesion and low coupling in the application.
- *Good UI design*:
    - *Ease of use*: UI should be clear and easy to understand for the user, thus overcrowding one activity with multiple features could lead to confusion.
    - *Visual Effects*: Visual effects should point out features and guide the user.

The Coding and Commenting Style in this application follows the *Google Coding Style for Java*. The naming conventions for this application are as follows:

- *Pascalcase* – for Classes and Activities.
- *Camelcase* – for methods and variables used inside classes.
- *Snakecase* – for variables used between classes.

Application Breakdown

This Chapter will go the main modules of the application, explaining their functionality, interface, responsibility, how they are used and interact. Listeners and principal methods are mentioned, depending on if a module makes any use of them.

CompassActivity

This is the main and default activity of the application. This activity allows the following functions: use the compass, use the camera, take pictures, save pictures, navigate to gallery or battery activities.
This activity is responsible for the following:

- Initializing the Compass & UI classes.
- Listening to the Compass Class whenever a new compass value is calculated.
- Inform user if the senor values are inaccurate.
- Inform the Compass Class on pause, start or stop of sensors.
- Updating the UI Class & UI Thread whenever the UI needs updating.
- Checking and asking the user for permissions only when required.
- Starting the Camera, taking pictures and saving them to public directory so they can be accessed from the public gallery.

This class does not have an interface, as all its methods are used within it and are made private in order prevent miss-usage. The only public methods used either @Override existing methods or communicate with other internally declared classes: Compass and UI class.

Compass Class

This class implements the Observer Design Pattern for the CompassActivity Class, where it only notifies it whenever a new bearing is calculated. The Compass Class also implements SensorEventListener, as it monitors any change in the sensors used. This class contains all necessary logic and workflow to related to the sensors and calculating the orientation of the device.
This class is responsible for the following:

- Pick the sensors needed for the application and monitor any change in them.
- Alert user if sensors are missing.
- Extract sensor values and convert them to orientation in degrees.
- Report new sensor value and its accuracy to the main Activity.

The Compass Class consists of private methods used to perform all the requirements to get and interpret sensor values, and an interface made of 2 methods:

- setup(Context c) – to initialize the class.
- onNewAzimuth(int v, int acc) – to report new orientation value and accuracy.

UI Class

This class is used to update the UI of the device that runs on a separate thread. The class has the following responsibilities:

- Update the compass image displayed to the correct orientation.
- Update the orientation text displayed with the correct value and direction.

The specific methods for updating the text and image are private, to prevent miss-use and allow anyone else to replace them if needed. The interface for this class consists of 2 public methods:

- setup(Activity a) – for initializing the class and its variables.
- updateCompassUI(int v) – used to update the class and UI.

Gallery Activity

This activity is used to display pictures taken with this application on the device. It consists of 3 elements:

- Gallery Button: opening the public gallery of the device allowing the user to pick any picture.
- ImageView: where the picture is displayed.
- TextView: displaying the bearing and direction when the image was taken.

The Gallery Activity has no interface as all its methods are private and used internally, and its only responsibility is to display chosen pictures from the gallery along with their bearing and direction.

Battery Activity

This activity is used to display the battery information of the device. It displays all the available public information that the device provides, without having to ask for further permissions from the user. This information is displayed on a TextView.

Sensors Used

The application uses the sensors only while in the main CompassActivity, otherwise they are on placed on pause or stopped. The application by default will use the Rotation_Vector sensor, as it provides better precision than other motion sensor; however, if the Rotation_Vector sensor is unavailable, the app will try use the Accelerometer and Magnetometer sensors instead; it these are unavailable as well, the app will alert the user that the compass feature of the app cannot be used. If the accuracy of the sensors is low, the app will inform the user about it and ask to move the device around in the shape of ∞.

Extra Features

The extra feature in this application is the Gallery Activity, and its implementation was explained in the Application Breakdown. Thea reason for adding this feature was because it would make sense for the user to want to inspect pictures taken with the app and the bearing, they were taken at.

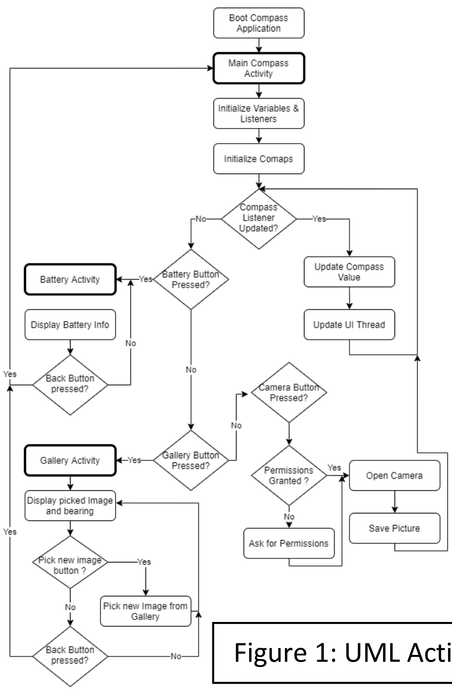

Figure 1: UML Activity Diagram

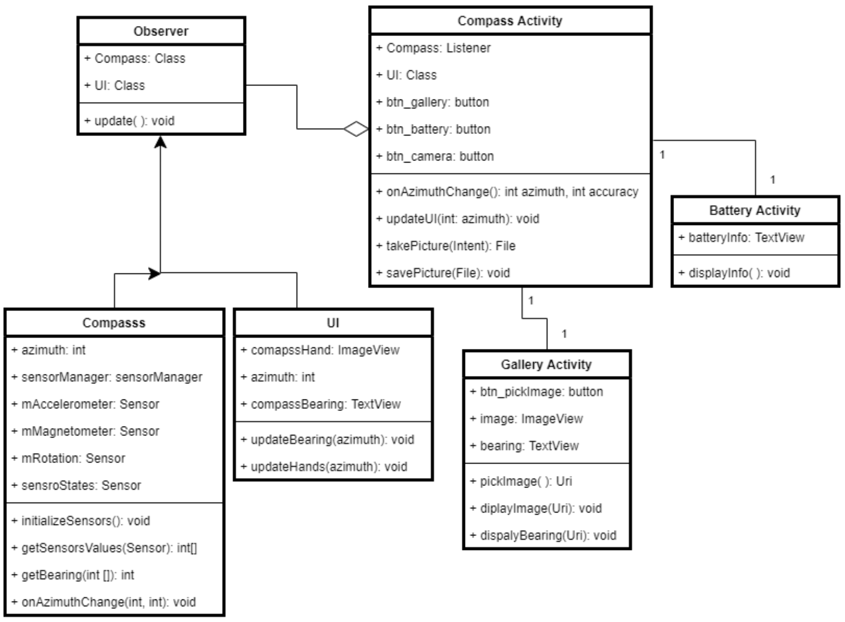

Figure 2: UML Class Diagram

References

[1] Camera API

https://developer.android.com/guide/topics/media/camera

[2] Rotation Vector Senor

https://stackoverflow.com/questions/14032825/whats-the-conceptual-difference-between-rotation-vector-sensor-and-orientation

[3] Coding Practices

https://source.android.com/setup/contribute/code-style

[4] Software Design Patterns

https://refactoring.guru/design-patterns