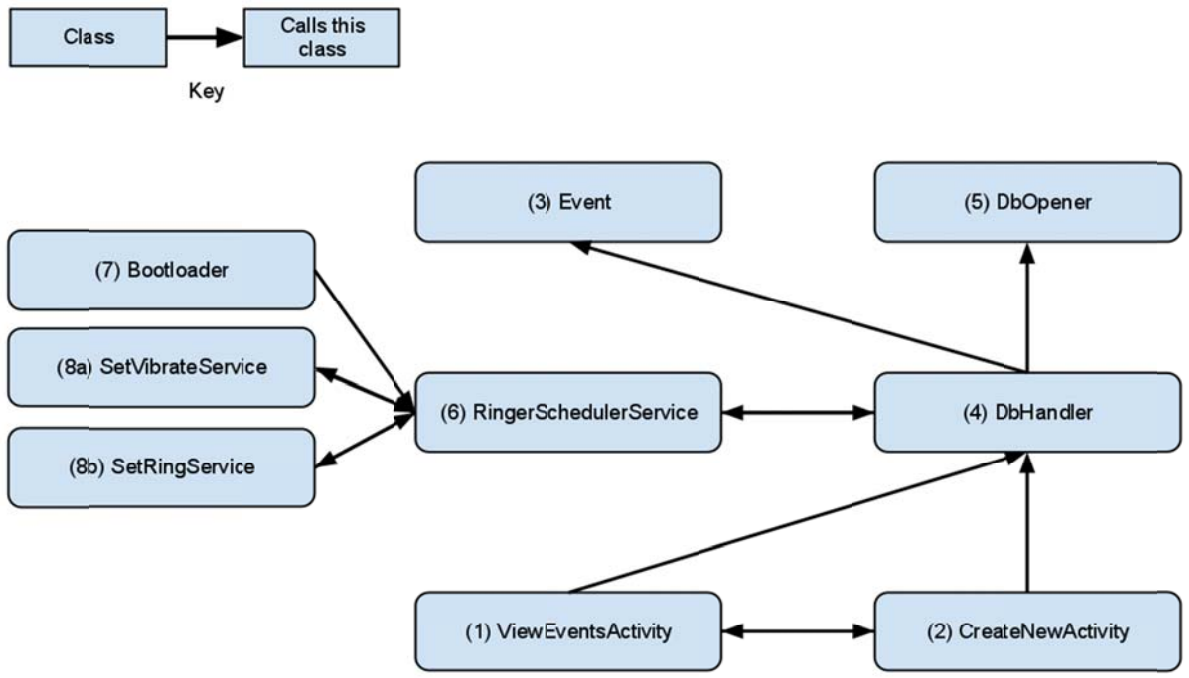
Ian Kenley December 10, 2011

Final Report: SmartRinger

Below is a flow chart of the component classes that make up the SmartRinger package, showing how the various classes interact with each other.



(1): ViewEventsActivity is the main GUI Activity that the end user can interact with. It uses a ScrollView nested inside of a RelativeLayout to prevent crashes in the case where the number of Events in a schedule exceeds the screen size. This allows a fairly basic GUI set-up to be robust in spite of a potentially large, varying schedule.

(2): CreateNewActivity is called when an end user presses the "Add New Event" button on the ViewEventsActivity screen. This Activity is just a series of input widgets that allow a user to, as the name of the class implies, insert a new Event into the database.

(3): An Event serves as both a container class for all the relevant data associating with a particular event as well as being a TextView itself. In extending Textview, it allows for end user interaction, such as deleting an Event by pressing it for more than 2 seconds when displaying on ViewEventsActivity. One **challenge** in having Event be a GUI wrapper and a container class for important data is that, since it is called from other classes that don't necessarily need the GUI

portion, the handling of Android-specific system data (that GUI classes must have) such as Contexts became tricky. In hindsight, it would have been better to decouple these, as it required passing Context references that were unnecessary much of the time, and presumably hogged resources.

(4) DbHandler is mostly a convenience class that allowed the other classes access to the SQLite database that allowed for persistent storage of the Events. While the actual debugging to verify the database was set up correctly was somewhat tricky, non-ADB issues were relatively minor. Since few of the queries allowed for user ~~injection~~ input, the basic CRUD interface is mostly hardcoded in to allow the other classes simple access to the underlying database. Though not currently set up as a ContentProvider (no outside programs use SmartRinger) it would not be terribly difficult to extend DbHandler into one.

(5) DbOpener is simply an SQLiteOpenHelper that is always called by DbHandler. It merely opens the database if it exists or, failing that, creates one.

(6) By far the most difficult from a design standpoint, the RingerSchedulerService handles the scheduling of the ringer settings based on Events. Figuring out how to do this was probably the **challenge** of designing SmartRinger. The first version simply pulled all of the Event data from the database every minute, which in addition to being inefficient was also error-prone. The second revision involved pulling the data once and then delaying each Event using the Handler class, which allows you to queue any class that implements the Runnable interface to run at a given time. What proved to be the downfall of this strategy was that the Handler class only accepts the time parameter based on the uptimeMillis and elapsedRealTime which are both based on time since boot. Trying to incorporate human times used in the Events into offsets proved to lack the robustness that any user would require (What happens on daylight savings? What if they change time zones?) . Fortunately, this **challenge** was eliminated by using an AlarmManager, which allows you to set a PendingIntent (essentially just a delayed activation of any of the main Android system classes) based on the standard system clock. Instead of scheduling all upcoming Events, RingerSchedulerService merely computes the nearest upcoming event and then sets a PendingIntent for the appropriate ringer setting service. This service is then called after the ringer settings are changed, the database is updated, or the system is rebooted.

(7) Bootloader merely catches a BroadcastReceiver message for the system booting up, and starts the RingerSchedulerService.

(8a/b) As their names imply, both SetVibrateService and SetRingService simply change the ringer settings. They are always called as a PendingIntent from RingerSchedulerService, and subsequently call it after they have finished their task to recalculate the nearest upcoming event.