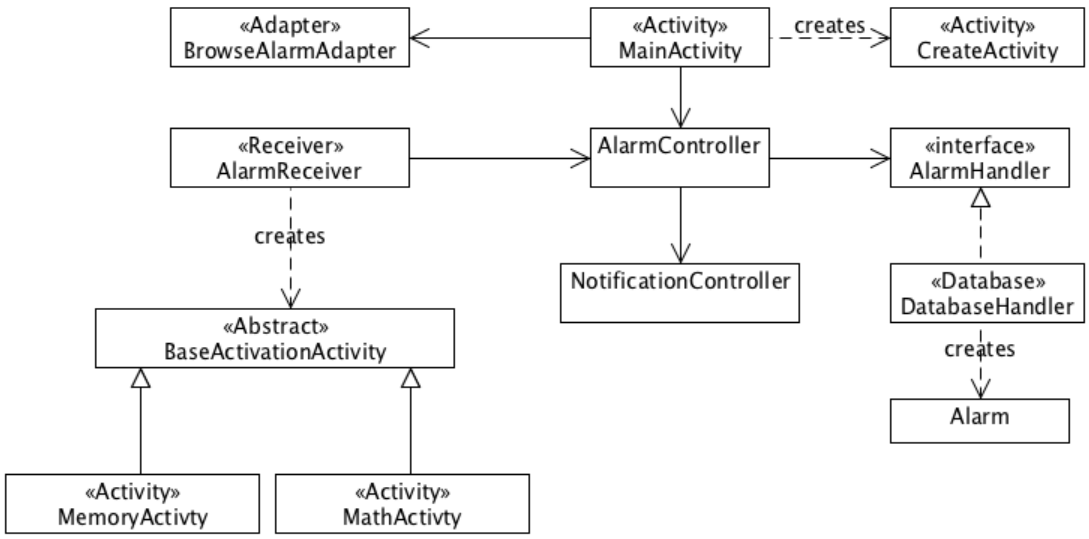
Architecture Specification

Our system is basically two systems. One part being the Alarm and the system that handles all the alarms and the other the modules which is started when the alarm is activated. The alarm part contains a database, notifications and views to show and edit alarms. The module part contains different modules with their own activity, model and controller.

System Design Overview:

General:

The system is designed after the MVC pattern, however with small modifications to work for android. Before explaining more about the system design to make it easier to read. We see an activity as a controller with regard to the model-view-controller. Throughout the system we use an Activity for handling the view and user interaction and then another controller responsible for the logic and modifying model objects. The alarm controller is a wrapper around androids AlarmManager and for more information about its functionality see Alarm Handling.



Modules:

To easily be able to create and switch between different tasks we have used a modular approach. A module is defined by subclassing an abstract activity. The activity that extends the abstract class will be the main controller of the module and also responsible for interacting with the user and making sure that the view is updated correctly after the model objects. The creation of the modules is made in a Module Factory. The modules inner design is independent

to the specific module and for more description read below. However they use the same package structure convention as the rest of the project.

Math Module

The math module consists of two controller one is the activity that is responsible for the view and and reading input from the user. The second controller is working as an access point to the model. The second controller is keeping track of the users progress, validating answers to problems and also for creating new problems.

The problems are randomly created by a factory class which randomly creates a new MathProblemType object. It is then up to the MathController to use the created problem and via it validate the users answer and generating numbers custom made for that problem.

MathProblemType is an abstract class that problem types must extend. The MathProblemType implements one method and then via the Template Pattern delegate the behaviour needed in the method to the implementing class. This is used since the problem types classes is responsible for generating numbers for their own problem based on a specified difficulty. The abstract class is responsible for based on the difficulty mapping it to the correct method for generating for example numbers to an easy problem.

Memory Module

The Memory Module differentiates it from the Math Module since it requires more ui and logic to handle user interaction. To keep the code testable we have chosen to separate the logic and ui code as much as possible. Therefore we have an Activity/Controller which responsible for user input and updating the view. It is also responsible for handling the timer for when two cards has been pressed. By doing this have all the game logic/rules in a separate controller, Memory Controller. Making it very easy to test the game logic independent of user interaction. We have done the same thing with the card object. We have on object for handling logic and the games state and then an wrapper around it for handling ui related code. This separation is necessary for being able to write the logic testable.

Software decomposition

The package structure follows class types, so activities pertaining to the main application are part of the .activity package, controllers in the .controller package and alarm activation modules in the .modules package, with their own module-specific .activity, .model, etc. packages as required.

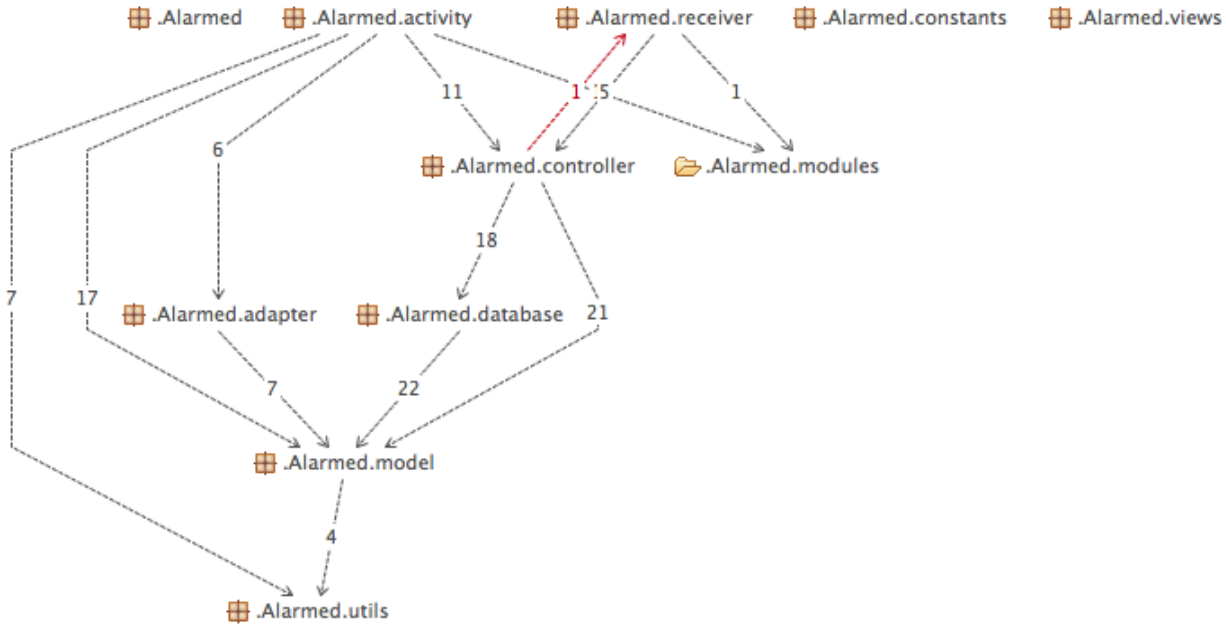
Dependency Analysis

General:

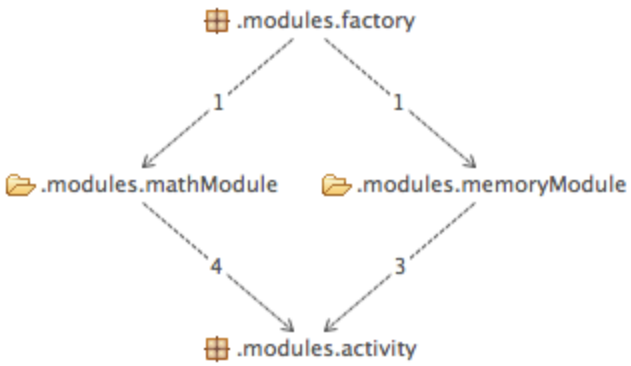
We got a circular dependency between AlarmController and AlarmReceiver. The receiver needs a dependency to the controller to be able to let the controller know that an alarm has been fired so it can set a new alarm. The problem is that the controller is deciding which receiver that should receive the alarm, so it needs to know the existence of the receiver.

We have tried to the connection to the receiver into a factory pattern, but that only moved the problem. You could also change the the connection into an observable pattern, but that only work if the controller is active. Our application will be inactive most of the time and only when the alarm is fired the receiver is activated. Then if the rest of the application is inactive, there is no controller and none to respond to the changed state of the receiver. This is also due to that android can without a warning terminate our application, since the android multitasking implementation.

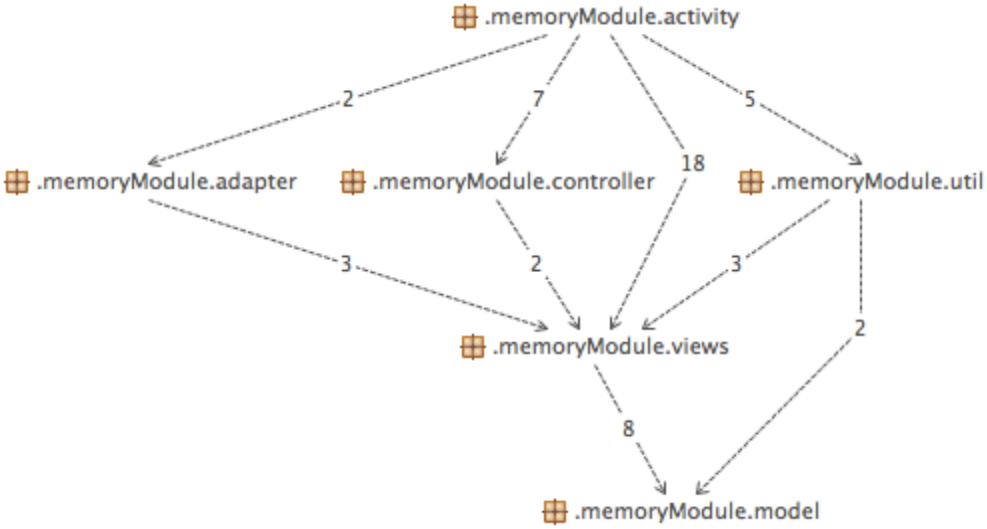
For dependencies, see below. Each subpackage got their own dependency analysis schedule.



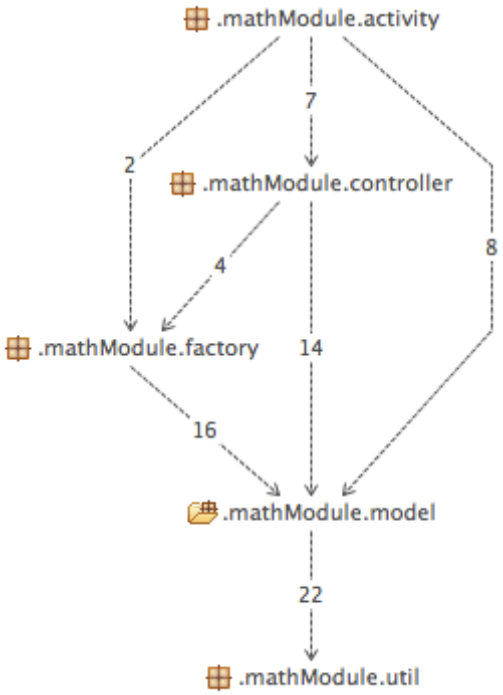
Modules:



Memory Module:



Math Module:



Persistent data management

Alarms will be stored in a SQLite database on the phone. The keys we are using are listed below, it's basically all you need to know about the alarm. Everytime the alarms get's updated the database called and returns the current state which all alarms are in.

Alarm:

- _id int (key)
- time text
- daysofweek integer
- enabled boolean
- module text
- volume int
- alarmtone text
- vibration boolean

Alarm Handling

Alarms that has been set is handled by the AlarmController. The MainActivity is responsible for the instance of the AlarmController when the application is running normally and no alarms are triggered. When an alarm is triggered an instance of the AlarmController is kept by the

AlarmReceiver for it to be able to notify the AlarmController that it has to update.

The controller uses the database to store the alarms through an AlarmHandler-interface, to be able to change how to store the alarms easily. It separates the how we store the data from the rest of the application. This enabled us to defer the development of the application data storage and makes it easier down the road to switch how the alarms is stored. For example if we decide to use another database solution, store using a filesystem or to save the alarm in the cloud.

It uses the default AlarmManager service in Android to schedule alarms. Whenever an alarm is created, activated, deleted, edited or enabled/disabled, the controller is rescheduling the alarm to the first enabled alarm and then tells the NotificationController to update the current notification.

References:

Android Activities Lifecycle: http://developer.android.com/guide/components/activities.html
MVC: http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller
Factory Pattern: http://en.wikipedia.org/wiki/Factory_method_pattern
Observer Pattern: http://en.wikipedia.org/wiki/Observer_pattern
Template Pattern: http://en.wikipedia.org/wiki/Template_method_pattern
SQLite: http://www.sqlite.org/