Developer Manual

For "Alarmed+"

Installation and setup:

1. Clone the git repository from https://github.com/rekoil/DAT255-EpiClock
2. Make sure you have the Android SDK (API level 15 and support library), an Android (Virtual) Device and Java 6 SE development environment.
3. From eclipse import the project as an android project. This will import the main project as well as the test project into your eclipse workspace.

Ant installation:

If you have recently cloned the project from the github repository you have to set up ant for your project. Assuming you have ant installed on your computer, if not visit the ant homepage at http://ant.apache.org/ and set it up to get going.

When standing in your project root run this following command to set up your local properties:

android update project -p .

And then when standing in the test folder AlarmedTestTest run:

android update test-project -p . -m ..

And you are ready to go.

Build Server:

We are using an continuous integration server in this project located at: http://jenkins.pepparkakan.net/. The build server builds and runs the test project every time someone pushes an commit to github. It also runs Emma on every commit and generates an coverage report which are available on build summary page.

Tests:

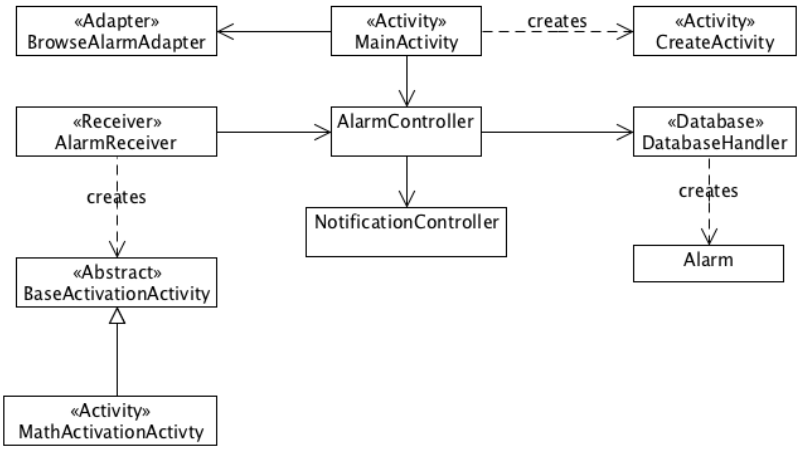
All the tests for the project are located in a separate test project, AlarmedTestTest. There are two ways to run the test suite.

- Eclipse
  - Right click on the test project and choose *Run As -> Android JUnit Test*
- Run the tests via ant (this requires the emulator)
  - Navigate to the test directory

    - Then run: **ant clean debug install test** (if the first time)
    - After that you can simply use **ant test**
- Via ant can you generate an emma test report via: **ant clean emma debug install test**
- Then open the AlarmedTestTest/bin/coverage/coverage.html in your web browser

Architecture:

Overall:
See figure below for an overall overview of the system design. See Modules for more information about the BaseActivation and module system.



Controllers:

The system is based around the AlarmController to set and acquire alarms from the AlarmHandler. The current implementation of the AlarmHandler is using an SQLite database to store the Alarms. The AlarmController is also responsible for scheduling the alarms via AlarmManager in Android. The second controller is NotificationController which creates an ongoing notification when an alarm has been created. It is responsible for managing the notification and updating and removing outdated notifications. It is also responsible for making sure that is always the alarm that is closest to trigger that is shown.

Package Structure:

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.

Modules:

Modules are stored in a separate package named .modules. Each module is only required to include an activity that can be launched by the application when an alarm goes off. Modules extend a ‘BasicActivationActivity’ which supplies them with methods for controlling the volume of the alarm and of course the ability to stop the alarm. All an implementing ModuleActivity has

to do is extend the the abstract activity and then do a **super.oncreate()** in the oncreate method and then when the user has completed the task **super.stopAlarm().**

Module Factory:
In order for the application to be able to launch an module, the module must be added to the ModuleFactory class. In the factory class simply add this line to the static block:

modules.add("ModuleName", ModuleActivity.class)

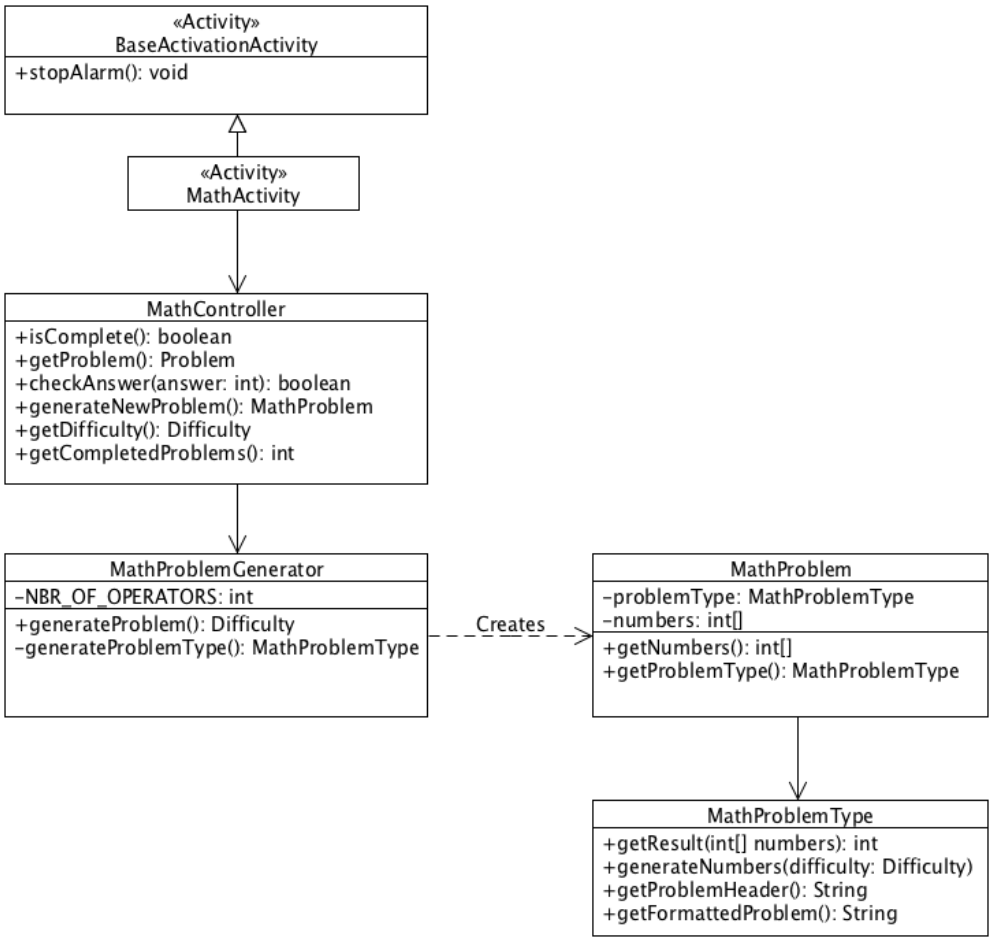
The rest is being taken care of by the application.

Module Guidelines
In the AndroidManifest.xml add this line for your module activity:
android:launchMode="singleTop"

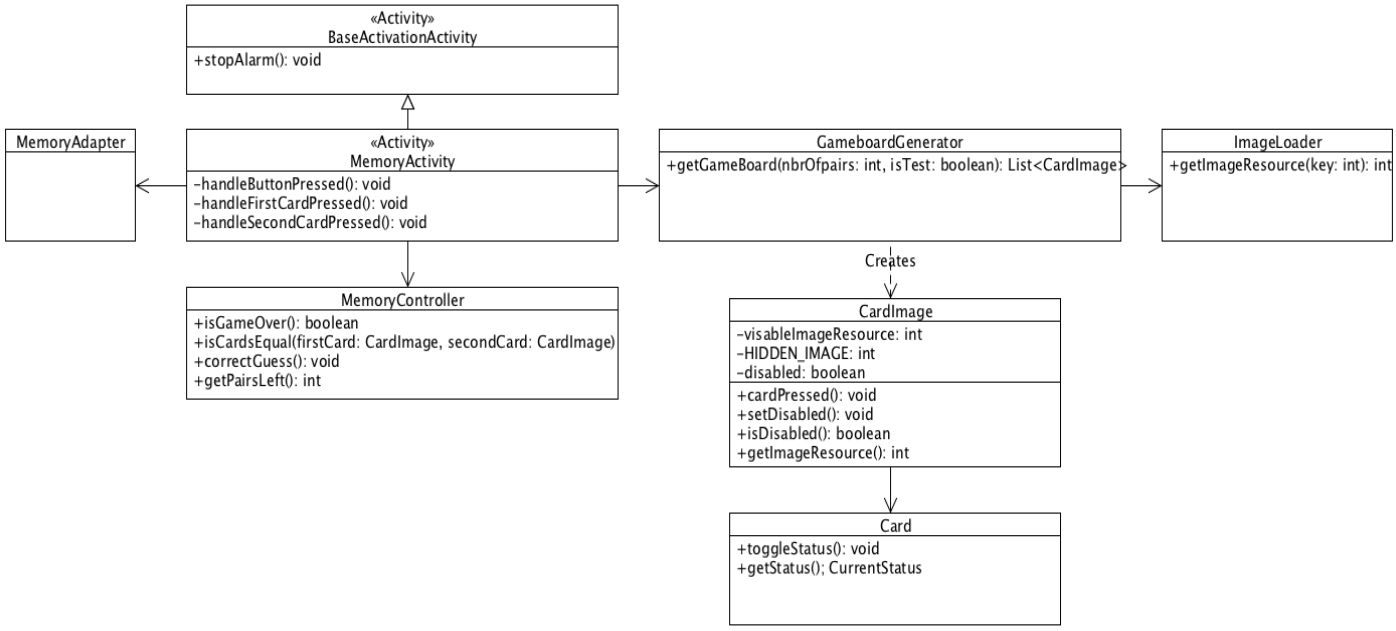
Math Module:
The Math Module currently has five different implementations of MathProblemType and is randomly selected and presented to the user.
They are:

- FibonacciProblem
- PrimeProblem
- AdditionProblem
- MultiplicationProblem
- FactorialProblem
- BaseSwitchProblem
- ModularProblem
- DifferenceOfTwoSquaresProblem

The MathController is responsible for generating new problems for the MathActivity, which is reading the input from the user and checking the answers with the controller, which in turn (based on the number of correct answers in a row) raises or lowers the difficulty.
For UML overview see the image below:



Memory Module:
The second module in the application is an simple implementation of the classic Memory game. If you are not familiar with how the game work, I can recommend reading up on wikipedia: [http://en.wikipedia.org/wiki/Concentration_(game).](http://en.wikipedia.org/wiki/Concentration_(game)) The MemoryActivity is responsible for handling the users input when it presses an card on the gameboard. The activity then uses the controller to check after the user has pressed two cards if they are equal. If they are equal the two cards are disabled. The controller is also responsible for keeping track of how many pairs that are left in the game. The CardImage class is a view wrapper for the Card class. All the imageresources constants is loaded via the image loader then via the GameboardGenerator that creates the gameboard and returns it to the activity. To display the CardImages to Activity uses an adapter.


«Activity»
BaseActivationActivity
+stopAlarm(): void
MemoryAdapter
«Activity»
MemoryActivity
-handleButtonPressed(): void
-handleFirstCardPressed(): void
-handleSecondCardPressed(): void
GameboardGenerator
+getGameBoard(nbrOfpairs: int, isTest: boolean): List<CardImage>
ImageLoader
+getImageResource(key: int): int
MemoryController
+isGameOver(): boolean
+isCardsEqual(firstCard: CardImage, secondCard: CardImage)
+correctGuess(): void
+getPairsLeft(): int
Creates
CardImage
-visableImageResource: int
-HIDDEN_IMAGE: int
-disabled: boolean
+cardPressed(): void
+setDisabled(): void
+isDisabled(): boolean
+getImageResource(): int
Card
+toggleStatus(): void
+getStatus(); CurrentStatus