ESE - Review of Team 1

By Andi Hohler, Jan Binzegger, Marc Dojtschinov, Sándor Török

Autumn Semester 2013

1. Introduction and General Design

At a first look the user Interface makes a sound and intuitive impression. Navigating in the Application works fine only one bug could be found (Look at section 7.) beside of that no other Bugs could have been found.The code design is overall very well structured. Each feature is separated in packages. making it open for extensions.
They separated the models representation of information from user interacting with it. Hence no violation of the MVC- pattern could be found.They used a lot of helper Objects for example they use a SportsListAdapter to add the the Data retrieved from the SQL-lite Database to the ModelView. In Addition they used further design patterns e.g. They make a good use of the messaging pattern to handle their views(ch.unibe.sport.network.message.class). This is a very nice way of building views.
The responsibilities are quite clear and there is in general only one main responsibility per class. Where they used Invariants they are of good use. To wrap up concerning the design there is in our view nothing particular to criticize. Therefore In our opinion this App is almost an example of writing good code. But still we found some details which could be done better or otherwise which will be discussed in this review.

1.1 Missing Features and improving performance

Most use cases described in the Software Requirement Specification of this project are well implemented however there are still some points missing:

The Use case “Share schedule” (Section 10 Software Requirements Specification) is not yet implemented. This should be done in the near future(we guess they already planned it)

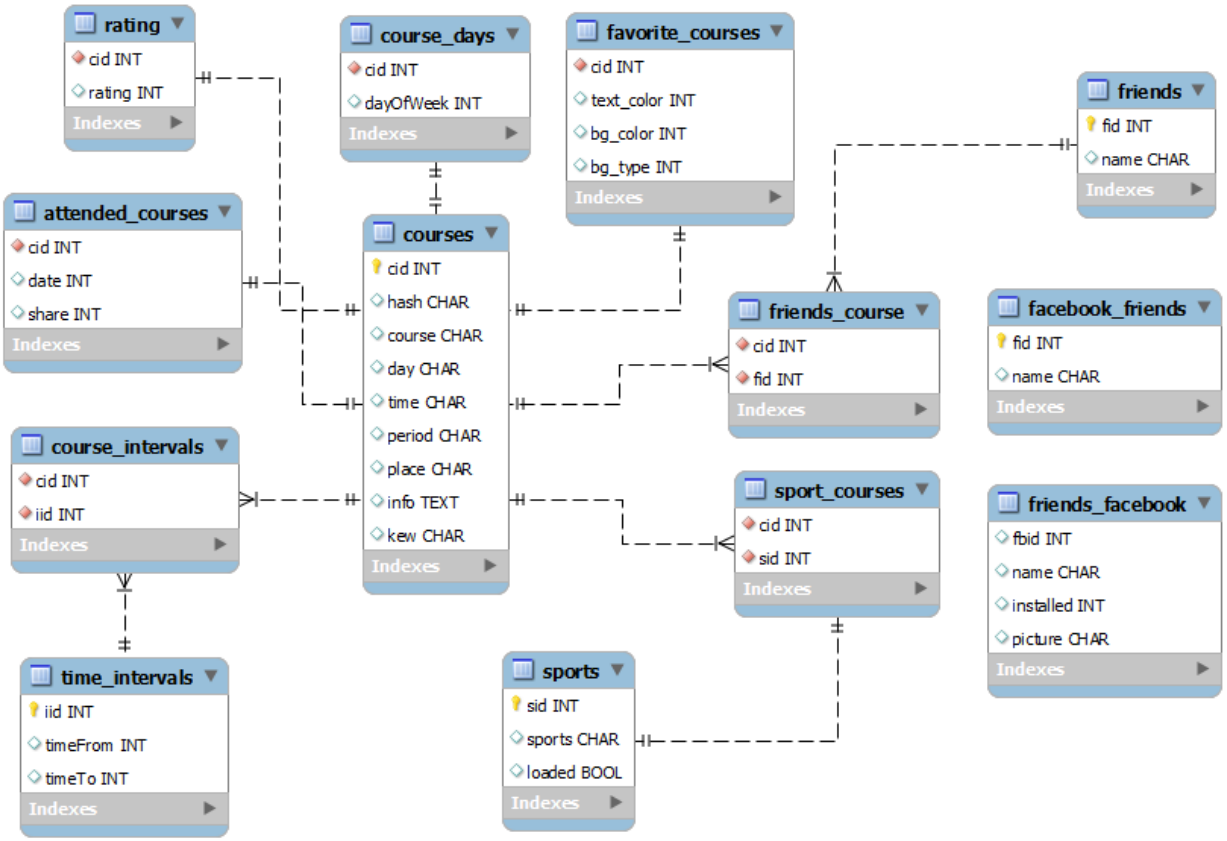
Following points we think would help to improve the user experience:

- Removing a scheduled course should be possible. At the current version we could not figure out how to remove a scheduled sport appointment.
- There should be an overview calendar showing all scheduled appointments. Right now you can only see if you scheduled a course by clicking on the specific course. We Suggest it would be great if you click on attend there could also be an entry in the phones base calendar.
- If you Rate a given Sport it take quite some time. Here it would be good to improve performance. The same with clicking on attend at a given course it seems like this should not take as long as it does.

2. Data Persistence

The main data is persisted within a SQLiteDatabase. It's done through a DBAdapter. At the same time we have classes for the tables and for each task. Additionaly preferences are saved as Key-Value Sets. There is also data retrieved through http requests.

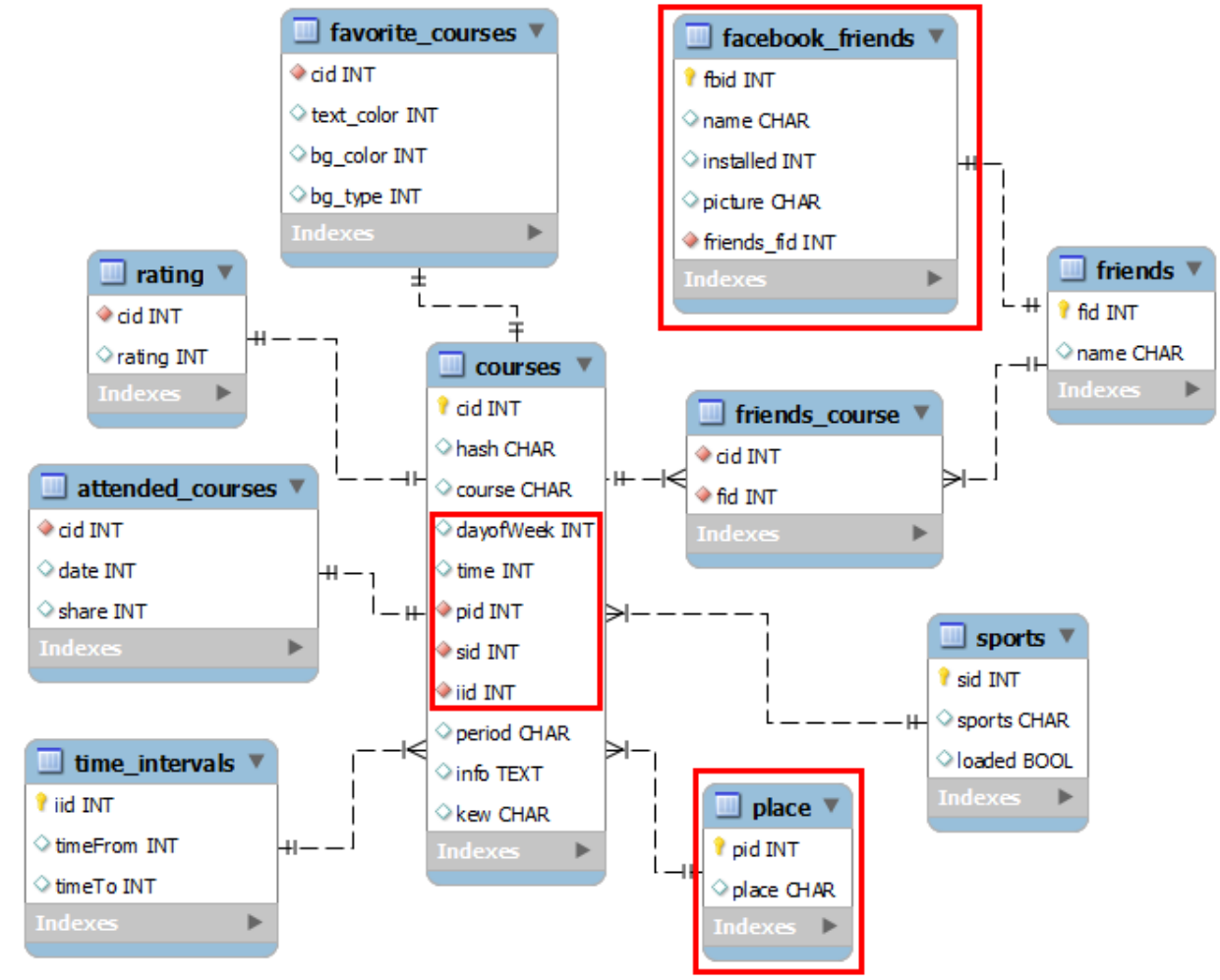
2.1 Database structure



Database entity relationship Diagram © Bugius

Considering the structure of the SQLite Database we can see the main table is the course. This Table is actually a bit overloaded and could be reduced. Also the course_days table seems strange and actually forms a redundancy to the courses table. As an improvement i would suggest the following changes:
First take the place out of the course, this will simplify the table because we have no redundancy of two courses having the same place. Also we are able to add GPS coordinates to the place table and make finding places better. This is actually a bug in this version. I was told to go to Basel for Badminton.
Secondly there are lots of join tables which in my opinion are not needed. Putting the id into the courses table is enough.
As a third the Facebook tables seem unfinished so just a proposal from my side how to connect

them. Assuming no one has more than one facebook account.
Additional you should do one standard of time. Either you use CHAR or INT which I recommend. Considering days of the week, just use the int and you can use a list for the Strings, this also makes multiple languages easier realisable.
Last but not least I would recommend to change database table names. Choose whether you want to us all in singular or in plural form.



Database entity relationship Diagram © Bugius

2.2 Key-Value Sets usage

Preferences seem well organized and named. Also Private Mode is used in my opinion these are fine.

2.3 Code organization

The package idea for the tables is good but the class CourseData seems wrong there.
Otherwise well organised and easy to get into the code.

3. Coding style

The overall complexity of the application code is very high I would say. The actual project contains 167 .java-Files with a total amount of 13119 lines of code that directly belongs to the app. There are many relations between classes.

Very good usage of the styles.xml with a standard base design and additional design elements for newer Android versions.
I don't know, if the facebook App-ID belongs in the strings.xml file, because it's rather a code-specific string than a public message which could be displayed to the user. As the facebook guide for developers mentions, the App-ID should be better stored directly in the AndroidManifest. (ref. https://developers.facebook.com/docs/android/login-with-facebook/)

It's really cool that you already translated the english strings file to russian (partly) and to german. Also all needed strings are stored in the strings.xml and never directly in the code itself.

Metrics
Plugin for metrics: http://sourceforge.net/projects/metrics/:

- *calendar.Day* both constructor have many arguments.
- *course.Course* constructor has many arguments
For both of this, you could consider using a specific Builder, but that's a detail. It could avoid confusion with the order of the parameters used in the constructor.

- *course.TimeParser* - the parse method is very long (74 lines)
- *DBAdapter.tasks* - the doInBackground method has 59 lines, perhaps a bit long.
- *info.HeaderView* - initBackground is a bit long with 50 lines, but regardless it's good readable
- *main.SportsListAdapter* - performFiltering could be complex

Metric	Total	Mean	Std. Dev.	Maxim...	Resource causing Maximum	Method
Number of Overridden Methods (avg/max per type)	0	0	0	0	/Sport@Unibe/src/ch/unibe/sport/course/TimeParser.java	
Number of Attributes (avg/max per type)	1	1	0	1	/Sport@Unibe/src/ch/unibe/sport/course/TimeParser.java	
Number of Children (avg/max per type)	0	0	0	0	/Sport@Unibe/src/ch/unibe/sport/course/TimeParser.java	
Number of Classes	1					
Method Lines of Code (avg/max per method)	88	29.333	31.962	74	/Sport@Unibe/src/ch/unibe/sport/course/TimeParser.java	parse
TimeParser	88	29.333	31.962	74	/Sport@Unibe/src/ch/unibe/sport/course/TimeParser.java	parse
parse	74					
TimeParser	13					
getIntervals	1					
Number of Methods (avg/max per type)	3	3	0	3	/Sport@Unibe/src/ch/unibe/sport/course/TimeParser.java	
Nested Block Depth (avg/max per method)		1.667	0.471	2	/Sport@Unibe/src/ch/unibe/sport/course/TimeParser.java	TimeParser
Depth of Inheritance Tree (avg/max per type)		1	0	1	/Sport@Unibe/src/ch/unibe/sport/course/TimeParser.java	
Number of Interfaces	0					
McCabe Cyclomatic Complexity (avg/max per method)		4.667	3.859	10	/Sport@Unibe/src/ch/unibe/sport/course/TimeParser.java	parse
Total Lines of Code	106					
Number of Parameters (avg/max per method)		0.667	0.471	1	/Sport@Unibe/src/ch/unibe/sport/course/TimeParser.java	TimeParser
Lack of Cohesion of Methods (avg/max per type)		0	0	0	/Sport@Unibe/src/ch/unibe/sport/course/TimeParser.java	
Number of Static Methods (avg/max per type)	0	0	0	0	/Sport@Unibe/src/ch/unibe/sport/course/TimeParser.java	
Specialization Index (avg/max per type)		0	0	0	/Sport@Unibe/src/ch/unibe/sport/course/TimeParser.java	
Weighted methods per Class (avg/max per type)	14	14	0	14	/Sport@Unibe/src/ch/unibe/sport/course/TimeParser.java	
Number of Static Attributes (avg/max per type)	6	6	0	6	/Sport@Unibe/src/ch/unibe/sport/course/TimeParser.java	

(Metrics of course.TimeParser, method lines chosen)

You could consider using some helper methods here, so the readability would be better.
These are more some really small details than important things, because the longest methods usually do some initialization or loading.

- *DBAdapter.tasks.CourseInfoLoaderTask* has a high cyclomatic complexity in *doInBackground*
- *DBAdapter.tasks.SportsListLoaderTask* has a high cyclomatic complexity in *doInBackground*
- *main.AnimationsViewPager* has a high cyclomatic complexity in *getAnimation*
- *main.SportsListAdapter* has a bit high nested block depth in *performFiltering*, but that's not a problem.
- *main.search.ActionBarSearchItem* has a high nested block depth in init [not important]
- *utils.Date* has a high cyclomatic complexity in *daysUntil* & *monthsUntil*, but these are only some calcuations.
- *utils.Json* has a high nested block depth in *buildAssociativeListofSubCourses*
- *layout.CollapseableLayout* has a high cyclomatic complexity in *onClick* & *onLongClick*
- *widget.view.SeekBar* - high cyclomatic complexity in *inTouchEvent*

(About cyclomatic complexity: http://en.wikipedia.org/wiki/Cyclomatic_complexity)

You use several good patterns in your code. Some that are given by the Android Developer Guide and some generic Java patterns.
- Listener, ViewHolder, AsyncTask, Observer, Top level and detail/edit views
- ActionBar, NavigationDrawer, Swipe views, Confirming and acknowledging
- and many others patterns have been used.

3.1 Consistency

- hashCode "prime" variable should everywhere be a prime number (not 8, 32 or so) (http://stackoverflow.com/a/113600 4. comment of this answer)
- partly missing Javadocs
- I miss good javadocs that describe the use of a specific class or method and their responsibilities. So I just could make an overview of the main functionality. But for this amount of code it's much work.
- *dialog.BaseDialog* has one unused import.

3.2 Intention-revealing names

Almost all methods have a good naming, I couldn't find any bad choice.

3.3 Do not repeat yourself

The methods are hold very generic and I didn't find any relevant redundant code

3.4 Exception, testing null values

Exceptions are raised and invariants test for null values.

3.5 Encapsulation

The data encapsulation is okay, but there are some classes that have public instance variables as already described. Normally, you should use accessor methods instead of public instance variables. But as you commented, the direct access results in better performance for Android JIT (Just-in-time compilation). Some research resulted in finding exactly that performance question on stackoverflow: http://stackoverflow.com/a/8876494

3.6 Assertion, contracts, invariant checks

Very good usage of design by contract. All necessary conditions get checked in invariant methods in all classes that have important instance variables.

course.Interval does not contain any null/invariant checks. Instance variable of time could be null. isUnknown-method could raise a NullPointerException.
You should check all relevant classes that get data from outside, if there are assertions.
The missing javadocs of methods could specify the contract between the involved methods, so it would be easier to see, if for example the parameter “id” must be unique, or can be negative or things like that.

3.7 Utility methods

- Very nice, that almost every used object class has a copy method.

4. Tests

There are 11 test suits and a total amount of 54 tests, which all are green. But 9 tests are empty and don't test anything.

4.1 CourseTest

- testInit, testSave, testUpdate don't contain any code. They should be filled with useful actions.

4.2 DayParserTest

- Checks if the DayParser works correctly and can handle the different input formats. Looks fine, enough tests for all different cases.

4.3 IntervalTest

- testGetTimeFrom, testGetTimeTo, testSetTimeFrom, testSetTimeTo (all tests) don't contain any code. They should test something.

SportTest:

- testInit doesn't contain any code and doesn't test anything

4.4 TimeParserTest

- contains a good amount of tests for this TimeParser class.
- several tests prove that the parser class can handle different input formats.
- there is also a test that handles a special case, where the time isn't set and needs to be arranged

4.5 TimeTest

- this test case tests the Time class.
- the Time class takes a specific amount of minutes as input in the constructor and converts it to hours and minutes
- the test coverage isn't as high as I would prefer. There is only one of the fourTime constructors tested
- also the nextHour, toMinutes, toString, copy & equals methods don't get tested.

4.6 UnknownTimeTest

- this test case doesn't contain any testing code but a empty testUnknownTime test.

4.7 AssociativeListTest

- sounds like a normal HashMap.
- tests the adding and valueOf methods.
- only covers add, get, getSize methods
- covers both of the two constructors

4.8 CalendarHelpTest

- tests the CalendarHelper
- this test set does test two different date classes: CalendarHelp and Date (not necessary bad, but also not very distinct)
- by the way: The Date class is very complex and long, 419 lines of code.
- allmost all public methods of CalendarHelper are covered by tests.

4.9 ColorTest

- tests only the Color class (good distinction)
- coverage is 50%, because the converting methods are not used in the tests.

4.10 DateTest

- the real Date class test case (check CalendarHelpTest)
- is a very long and well crafted test case for this class
- the coverage is high, the tests easy to understand

In general, the tests are nicely readable, the test name contains good enough information and the test classes are distinct. The testing code is easy to understand. The test coverage is not everywhere as good as expected. Some test cases only contain empty tests, what is a bit strange.
The set of all tests isn't so big and just cover a small part of the actual code & functions.
I would first suggest to implement the empty tests and improve the coverage of the actual tests. After that, you can add new tests for the GUI (basically the Activities) itself.

5. Documentation

In general the comments made in the code are understandable and helpful furthermore where javadoc comments were used they are of good quality and they reveal their intentions. Overall there could be many more class comments describing the responsibilities of some classes to help people that are not familiar with the code understanding the purpose of this class quicker. On the other hand the team did a good job naming classes, methods and variables consistently and appropriate what makes a lot of comments obsolete.

6. Analyzing CourseInfoActivity

Overall the Activity is well crafted, having good defined and relevant responsibilities. All the Activities extend the ProxySherlockFragmentActivity, which makes communication with messages easy in the whole application. Furthermore removes also redundant code for the actionbar customizing.

The CourseInfoActivity calls the parent class in its constructor with its string identifier, elegantly allowing logging without redundancy.

The process method reinitializes the course, sets the corresponding favorite button in the action bar, updates the view and the rating of the current course.

In the onCreate method the view is set up with the help of the handy Loader inner class. The Loader asynchronously inflates the needed views and notifies the activity when the task is completed. On completion the view is added to the container.

The show method basically starts the activity itself, allowing the launch with a single line:
CourseInfoActivity.show(Context ctx, int courseID);
It is not violating the Android guidelines for communication between activities, because an Intent is used to start it.

When favorite button is clicked in the action bar, the database is updated in the onOptionsItemSelected method, through an asynchronous task. On completion a message is sent, that this course should be updated.

The onKeyDown method finishes the activity with a delay of 150ms. In the comment the reasoning says, finishing activity without delay causes action bar in main activity to collapse action views. In the SportInfoActivity, which should be the previous activity in the back stack, the views are also collapsed, when the screen orientation is changed.
In my opinion this method is unnecessary and is not solving the above mentioned issue. I think the problem lies in the lifecycle handling of SportInfoActivity. More precisely the expanded views or their ID should be added to a bundle when onPause or the onStop method is called, then restored when the activity is recreated.

7. Found bugs and TBDs (informative and not closely related to this review)

- Android minimum SDK is set to 10, but the app is crashing on some older devices.
  For example:
  - navigation drawer click on about
  - search click on the arrow
  - click on a course in one sport
  I (Andi) contacted Aliaksei from Team 1 to debug this problem. The problem was really, that my device was too old and didn't know about the biggest screen resolution that could be available.
  So in the xml-file was defined, that a Drawable should be created from card_bg. The card_bg.xml defines the background itself. The png-file for the background was located in the xxhdpi folder as card_bg.9.png. As my device doesn't know xxhdpi resolution, it couldn' find the .png file and just reopened the card_bg.xml. This resulted in an unlimited loop what caused a StackOverflowError. Aliaksei simply changed the png filename to card_bg_white.9.png and all worked. He then provided me the fixed apk to test the other functions.

- In SportInfoActivity the expanded views are collapsed after screen orientation is changed

- In CourseInfoActivity landscape orientation nothing is shown (Android 4.1.2)

- Attended courses are always in favorite list

8. Conclusion

To sum up Team 1 did a very nice job. Their coding style is very clear and well structured except for minor details. They follow common principles and made a good use of patterns. Concerning the comments and documentation they followed a limited policy in our view they could add some more information about class responsibilities. But this is a matter of programming style and can be viewed differently.

As a point of improvement I would suggest to refactor the database structure. We strongly recommend NF3 or NF4 because not well structured database usually has a big impact on performance due to java being slow with SQL.