**Code Review Mensa@Unibe ESE 2013 Group7**

Reviewed By Group 4

Dominik Seliner, Domenico Iapello, Tim Kosadianos, Andreas Wächli

Assignement 1. 3

Design 3

Violation of MVC Layers 3

Usage of helper objects between view and model 3

Rich OO domain model 3

Clear responsibilities 3

Sound invariants 3

Overall code organization & reuse, e.g views 3

Coding style 3

Consistency 3

Intention-revealing names 3

Do not repeat yourself 4

Exception, testing null values 4

Encapsulation 4

Assertion, contracts, invariant checks 4

Utility methods 4

Documentation 4

SRS-Document 4

Understandable 4

Intention-revealing 4

Describe responsibilities 5

Match a consistent domain vocabulary 5

Test 5

Clear and distinct test cases 5

Number/coverage of test cases 5

Easy to understand the case that is tested 5

Well crafted set of test data 5

Readability 5

Assignement 2. 5

Assignement 3. 6

Mensa Activity 6

# Assignement 1.

Write a constructive and useful review of the project assigned to your team using the checklist template reported below. The review should serve as guideline to improve both code and design.

## Design

### Violation of MVC Layers

The usage of “Model-View-Controller” as a software architectural pattern, which separates the representation of information from the interaction whit it, is clearly realised over the entire project. In general the pattern is provided by the architectural structure of an android application.

Clear structured source tree in “Model” and “Controller” packages so also the “Views” in the activity layout Folder.

### Usage of helper objects between view and model

Good usage of helper objects between view and model.

### Rich OO domain model

The code contains a lot of useful domain models. The problems attributes are clearly shown and modelled in the code. The Mensa object is built trough a builder in and implemented in proper OOP manner.

The SRS document could contain more relationships to the modelled domain objects (there are more domain model in the code as in the diagram).

### Clear responsibilities

The responsibilities in the code are separated clearly. There are many small classes containing a well distinguishable responsibility. There are no godclasses containing to many responsibilities and the majority of methods consists of less than 4 lines of code.

### Sound invariants

Invariants are checked inside the builder method and in many methods where it makes sense.

### Overall code organization & reuse, e.g views

Good code organisation. The code is not all in one package, so logically good structured.

## Coding style

### Consistency

The team has a consistent coding style.

### Intention-revealing names

Names are chosen very well. Most of the time the intention is clear, and otherwise the documentation resolves uncertainties.

### Do not repeat yourself

Sometimes Documentation only repeats what the method or variable names already imply.

### Exception, testing null values

When an exception is caught, there is a message in the log. The team also implemented testing of null values.

### Encapsulation

Encapsulation is achieved by private variables.

### Assertion, contracts, invariant checks

There are assertions in the code where needed. Invariants are checked

### Utility methods

The team implemented utility methods, where an instantiation of an object for a task wouldn’t make sense.

## Documentation

### SRS-Document

Missing a table of contents at the beginning of the Document.

The SRS-Document was kept up to date in a good manner.

The purpose is clearly defined and understandable.

Under The point Stakeholders is not clear from where does the knowledge comes, that 75% of students at the Unibe are android user?

The System overview contains all of the implemented features.

In general the use cases are very short but they contain the necessary information’s.

In point 2.2 what does it mean “average intelligence”.

Good overview of requirements, missing newest acquired knowledge about the app.

### Understandable

The java-doc are mostly present ant helpful. No presence of java-doc in the java activity`s. In some cases the documentation is redundant, mostly there is a redundancy between a @return statement and the overview or description of the method.

Example:

/\*\*

\* actually reads the whole file and returns it as string

\* @return the file content as a string

\*/

### Intention-revealing

The naming of variables, methods and classes is properly adapted to the persistent problem they are intended to solve. So in addition whit the javadoc the intention of classes and methods is apparent.

Some comments are missing in the android specific (activity related) java files. A presence of them would facilitate to understand long code sections.

### Describe responsibilities

Due to good javadoc and naming the responsibilities of all the non android specific classes is well described.

### Match a consistent domain vocabulary

The vocabulary used to describe the domain for the project is well chosen and matches to the domain.

## Test

### Clear and distinct test cases

There are focused on one class and it’s methods. Using mock objects, the team managed to let the results depend on the functionality of one class only.

### Number/coverage of test cases

Regarding the number of model classes, there aren’t many tests. But most untested classes there, depend on the mensa-API. Such classes may be hard to test. The tests that are there however, contain tests for most methods of the class (excluding getters and setters).

### Easy to understand the case that is tested

Clean setup of mockobjects and well separated tests.

### Well crafted set of test data

Each test situation is cleanly setup and tested. This includes mock object usage whenever interaction with another class is needed. Mocks for classes that are used in multiple tests, are setup in the setUp() method. The mocks that are only used once are defined inside the specific testmethod.

### Readability

Good readability is achieved by with good method names and documentation.

# Assignement 2.

Analyse and describe the strategy used to persist data. You can use a UML diagram to be clearer in your description. Split the adopted strategy in multiple steps and criticize the overall design. If a specific step of strategy is not well design, provide details on how it can be improved.

In the top level menu of the app, there is an if-statement that checks if there is data that needs to be requested from the mensa-api. This is the case, if the app is started at a given date for the first time or if the user clicks on refresh. In this case, all the information of all the mensas is saved into the local storage, just like the android documentation recommends. After this, all the data about mensas and menus is loaded from the internal storage.

# Assignement 3.

Pick an activity of choice and analyse its code. Does the class have too many responsibilities? Is there some logic that should be moved to another class? If so, why?

## Mensa Activity

The Mensa Activity implements activity lifecycle methods like onStart or onStop, and has no misplaced functionality. So the class don’t have to many responsibilities and there is no need to outsource code. The class already delegates responsibility to other classes like the mensaListFragment.