Code Review Team 9

Content

1. Overall Review

1.1 Design

The view and the model are not very well separated. The view does too much which would belong into (new) model classes. For example: when a new list is added by the user, the view creates the shopping list object and also orders the database to store it. This could be improved by having a model class which has a method “createNewShoppingList(listname)” which would create the object and also store it. At the moment, the view is responsible for both instantiating and storing a new ShoppingList, this is too much responsibility which is completely unrelated to the actual view. The responsibilities of a view class should only be to assure that the data is properly displayed and the user can navigate in the app. It’s not the responsibility of a view to save the data (except if the view needs to detect state changes like onPause to call some saving procedure).

It would also help to have the model in a separate package as this would make the distinction between model and view easier to see. Also the utility classes, e.g. the database handlers, should not directly display error messages (Toasts) if possible. With the current design it would be very hard to change the way errors are being displayed. Visual output should only be done in actual view classes or by a special class which only does show status messages.

The decision to have a model which “does nothing” (pure data classes) is questionable. This leads to some constructs which seem illogical, for example that the ShoppingList is not actually a List of Items (you use HashMap<ShoppingList, Item[]> instead of just adding the functionality of storing a list of Items in the ShoppingList). This means to add a new Item to an existing ShoppingList it would be necessary to call the database. As the domain model will always be very light weight in this project (i.e. no deferred loading for heavy resources like images needed and no problems holding everything in cpu memory) it could be an advantage to always hold a real model (which actually represents the domain model in a useful way) in memory and only

save from time to time or when the user exits the app.

Last but not least: don't put mock classes in your project package (MockDataBaseHandler), mock classes should be in the test project.

1.2 Coding style

The database handlers are often used in a very bad way by just creating new instances every time the database needs to be accessed. It's not the point of objects to use them just once, you could easily replace all the calls with static methods in that case. Also instantiation (and garbage collecting) is expensive and shouldn't be done if not needed. If a method needs to access the database, the corresponding handler(s) should just be instantiated once at the beginning of this method or even be stored as instance variables if the class uses the database a lot.

The instance variables in the Notification class should be private, also consider to always use accessor methods instead of public fields, even if they are final. For example the standard java string class, which is immutable, does that too (e.g. .length() instead of .length to access the length of a string).

There are some strings which need to be extracted to the string resource xml, especially all the strings used to display toasts. Also some strings used as keys to store information in intents should be extracted as constants (for example the ones used for the intent to start DisplayItemsActivity), same goes for strings used to store data on the Parse service.

Interesting are also some of the standard code metrics (use the eclipse plugin http://sourceforge.net/projects/metrics/ to calculete them for yourself):

Metric	Total	Mean	Std. Dev.	Maxim...	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/max per		1.6	1.099	8	/ShoppyApp/src/com/eseteam9/shoppyapp/NotificationAdapter.java	getView
Number of Parameters (avg/max per method)		1.361	1.158	7	/ShoppyApp/src/com/eseteam9/shoppyapp/Item.java	Item
Nested Block Depth (avg/max per method)		1.503	0.946	5	/ShoppyApp/src/com/eseteam9/shoppyapp/OnlineDatabaseHandler.java	getUser
Afferent Coupling (avg/max per packageFragn		8	0	8	/ShoppyApp/src/com/eseteam9/shoppyapp	
Efferent Coupling (avg/max per packageFragm		17	0	17	/ShoppyApp/src/com/eseteam9/shoppyapp	
Instability (avg/max per packageFragment)		0.68	0	0.68	/ShoppyApp/src/com/eseteam9/shoppyapp	
Abstractness (avg/max per packageFragment)		0.045	0	0.045	/ShoppyApp/src/com/eseteam9/shoppyapp	
Normalized Distance (avg/max per packageFra		0.275	0	0.275	/ShoppyApp/src/com/eseteam9/shoppyapp	
Depth of Inheritance Tree (avg/max per type)		2.636	1.333	6	/ShoppyApp/src/com/eseteam9/shoppyapp/MainActivity.java	
Weighted methods per Class (avg/max per typ	248	11.273	6.85	28	/ShoppyApp/src/com/eseteam9/shoppyapp/ItemHandler.java	
Number of Children (avg/max per type)	5	0.227	0.734	3	/ShoppyApp/src/com/eseteam9/shoppyapp/LocalDatabaseHandler.java	
Number of Overridden Methods (avg/max per	18	0.818	0.936	3	/ShoppyApp/src/com/eseteam9/shoppyapp/DisplayListsFragment.java	
Lack of Cohesion of Methods (avg/max per typ		0.206	0.235	0.571	/ShoppyApp/src/com/eseteam9/shoppyapp/Item.java	
Number of Attributes (avg/max per type)	43	1.955	1.846	7	/ShoppyApp/src/com/eseteam9/shoppyapp/Item.java	
Number of Static Attributes (avg/max per type	29	1.318	2.583	8	/ShoppyApp/src/com/eseteam9/shoppyapp/MockDataBaseHandler.java	
Number of Methods (avg/max per type)	144	6.545	4.387	15	/ShoppyApp/src/com/eseteam9/shoppyapp/ItemHandler.java	
Number of Static Methods (avg/max per type)	11	0.5	0.839	2	/ShoppyApp/src/com/eseteam9/shoppyapp/MockDataBaseHandler.java	
Specialization Index (avg/max per type)		0.546	0.619	1.667	/ShoppyApp/src/com/eseteam9/shoppyapp/WelcomeScreen.java	
Number of Classes (avg/max per packageFragr	22	22	0	22	/ShoppyApp/src/com/eseteam9/shoppyapp	
Number of Interfaces (avg/max per packageFra	0	0	0	0	/ShoppyApp/src/com/eseteam9/shoppyapp	
Number of Packages	1					
Total Lines of Code	1815					
Method Lines of Code (avg/max per method)	1149	7.413	7.098	36	/ShoppyApp/src/com/eseteam9/shoppyapp/DisplayItemsActivity.java	addDialog

You definitely have too many arguments in the constructor of the Item class (and also in the ShoppingList). Consider using a builder class here to avoid confusion with the order of parameters. Also your average lines of code per method is rather high. Especially in some of the view classes you could introduce more small helper methods to improve readability.

It's good that the cyclomatic complexity has a maximum of 8 (as 10 is usually the proposed maximum for this metric), this means you don't have overcomplicated methods, even if they are a bit long in average.

1.3 Documentation

The documentation is overall not very useful, as there are only class comments describing responsibilities and very few method comments. It would be good to describe the contracts of all public methods in the java doc comment. For example it is unclear what the contract for an id passed to an Item is. Does the id need to be unique or not? (as there is a constructor which initializes it to zero automatically one could think it doesn't really matter what the id is).
Also the class invariants should be specified in the class comments. The class comments alone do often not help much in understanding the code, especially if there are classes like ShoppingList (as mentioned before: ShoppingList is not actually a list).

1.4 Tests

Not all tests did run in version v1.0, you also have some tests which don't run because they are not "finished", e.g. MainActivityTest. This test does not run, not because your app is not there yet, but simply because the test doesn't seem to be finished (a stub from a tutorial?). Also the DatabaseTest is completely broken, e.g. you create an array of length 1 and try to access the 1st position (instead of 0st). Failing test cases should imply that the program does not satisfy the requirements yet, but in your case they simply fail because they contain some severe programming errors. Also the DummyTest could be deleted as it doesn't do anything useful.

You overall already have quite a large number of distinct test cases (38 test cases). It's good that you especially test the database related classes, as they are non-trivial classes which could be error-prone and it's usually difficult to detect and locate those errors by just using the app.

The test cases are usually well separated, but some test cases could be even more separated into smaller tests, e.g. equals methods could be tested in a more "scientific" way by splitting the test into smaller named test methods. This would improve the readability of the tests a little bit.
Also there are a few tests which don't really test anything, like testConstructor() in ItemHandlerTest or some test cases in the UserHandlerTest.

It's also bad practise to always create a new instance of ItemHandler in the ItemHandlerTest. Of course the ItemHandler does not really have a state, but it should be used in the tests like it would be used in any other part of your application, i.e. create an instance and then reuse it for one test case. Also the testGetUnboughtMethod is useless as the last line is commented out. Did this test go green by just eliminating the actual testing?
Overall it's obvious that the testing is all work in progress, but still it would have been very nice to have only finished tests for v1.0 (which means they run and actually test something).

1.5 Bugs

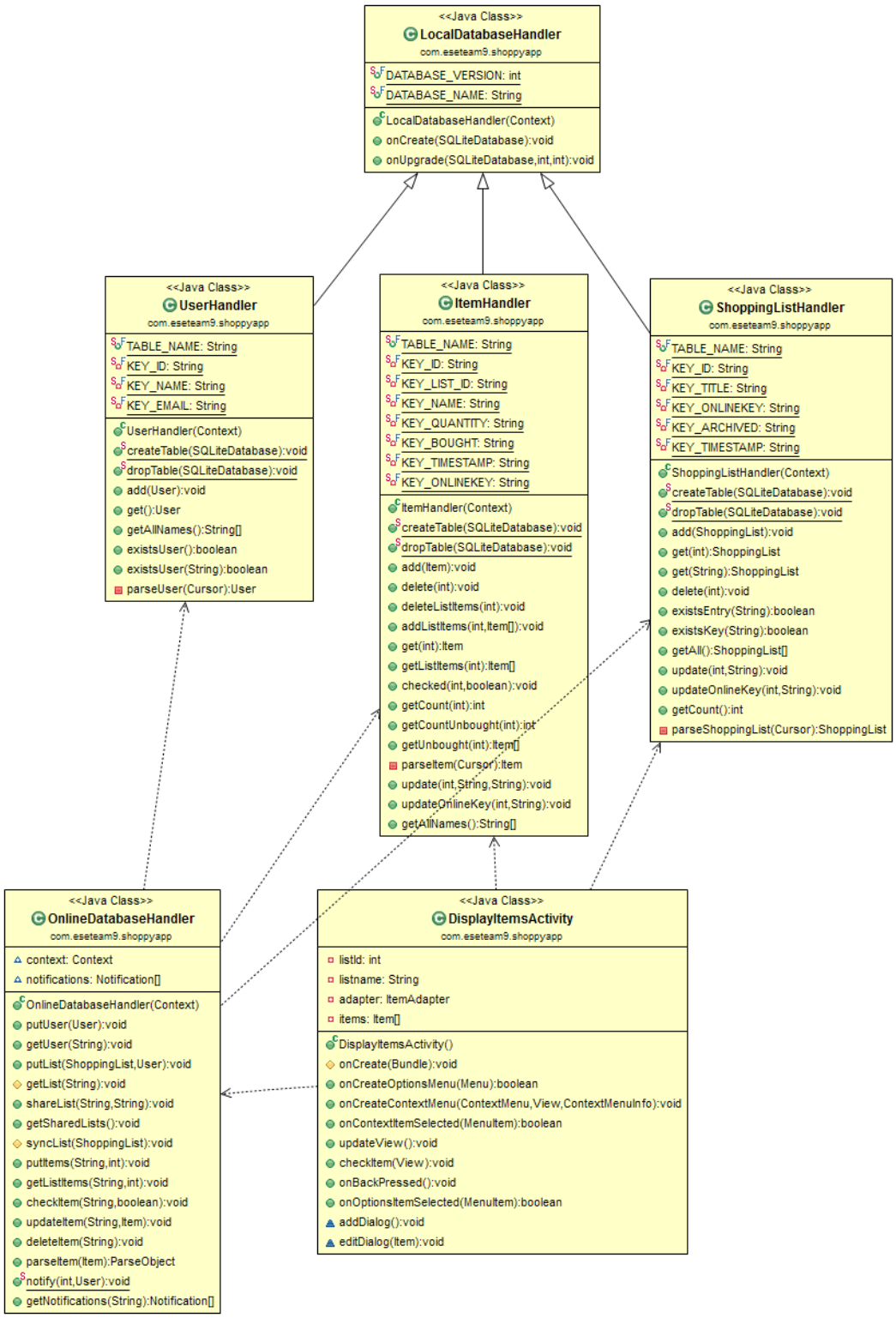
One bug found was the strange behaviour after adding the second item to a List. Afterwards a

whole lot of items get imported which seem to have some test purpose, but should not be in the final version.
Another Bug sometimes occurring is the instead of adding a new List, two new Lists “groceries” and “household articles” are added. It would have been good to have all “test data” removed from v1.0.

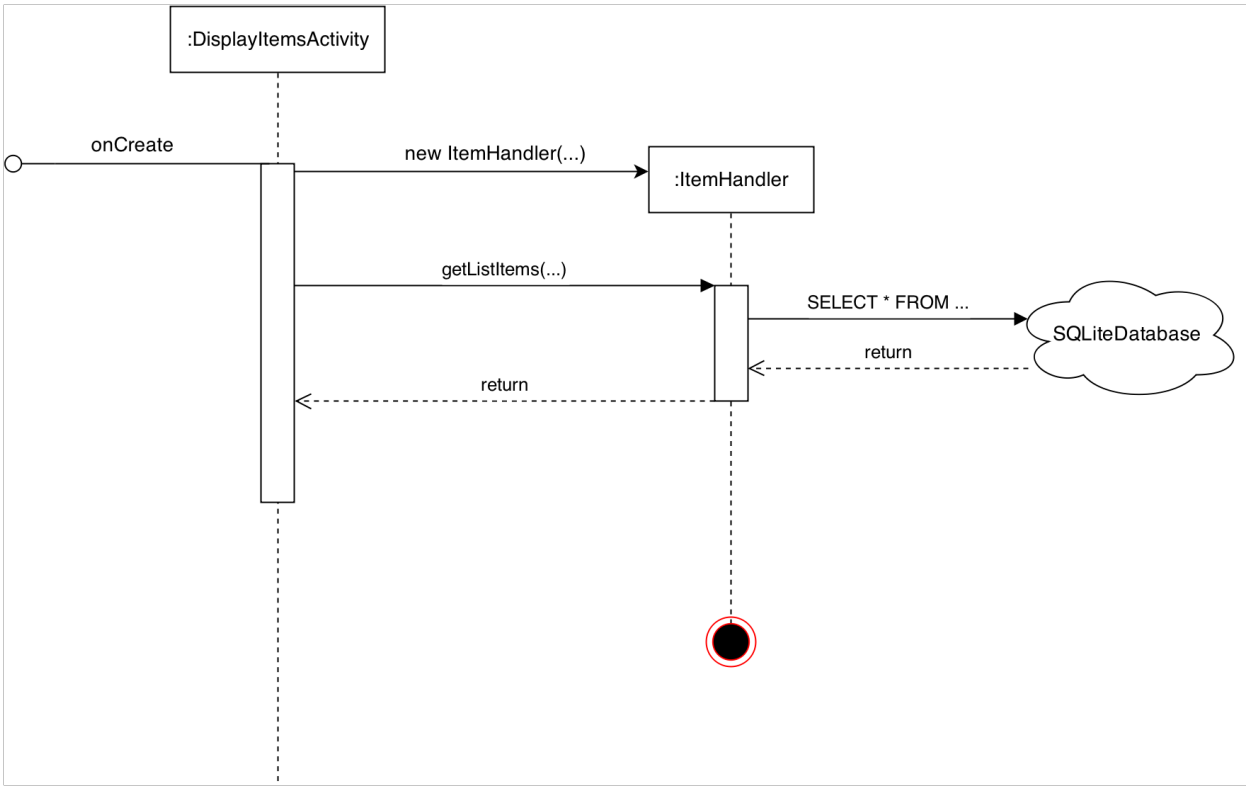
2. Persistant Data Management Review

The data that is only used locally, i.e. the lists, is saved by using the classes UserHandler, ItemHandler and ShoppingListHandler. Since these classes extend the class LocalDatabaseHandler which itself extends the SQLiteHelper, they use an SQLite Database on the device. Their content consists of methods to save and get data from the SQLite database.
Then there's the class OnlineDatabaseHandler whose responsibility is to save and get Data from an online database. It includes all methods needed for the data from users, items, lists of items and notifications.

The following uml class diagram also gives a short overview over the mentioned persistance management classes and shows the dependencies from the DisplayItemsActivity to the database handler. It is also very well visible how too many different classes depend on the individual database handler classes. E.g. the activity directly depends on multiple handlers, same as the online database handler. It would be a good idea to introduce an “LocalDataBaseHandler” with similar functionality as the online data base handler (and rename the current LocalDataBaseHandler to something like AbstractDatabaseHandler).


<<Java Class>>
LocalDatabaseHandler
com.eseteam9.shoppyapp
DATABASE_VERSION: int
DATABASE_NAME: String
LocalDatabaseHandler(Context)
onCreate(SQLiteDatabase):void
onUpgrade(SQLiteDatabase,int,int):void
<<Java Class>>
UserHandler
com.eseteam9.shoppyapp
TABLE_NAME: String
KEY_ID: String
KEY_NAME: String
KEY_EMAIL: String
UserHandler(Context)
createTable(SQLiteDatabase):void
dropTable(SQLiteDatabase):void
add(User):void
get():User
getAllNames():String[]
existsUser():boolean
existsUser(String):boolean
parseUser(Cursor):User
<<Java Class>>
ItemHandler
com.eseteam9.shoppyapp
TABLE_NAME: String
KEY_ID: String
KEY_LIST_ID: String
KEY_NAME: String
KEY_QUANTITY: String
KEY_BOUGHT: String
KEY_TIMESTAMP: String
KEY_ONLINEKEY: String
ItemHandler(Context)
createTable(SQLiteDatabase):void
dropTable(SQLiteDatabase):void
add(Item):void
delete(int):void
deleteListItems(int):void
addListItems(int,Item[]):void
get(int):Item
getListItems(int):Item[]
checked(int,boolean):void
getCount(int):int
getCountUnbought(int):int
getUnbought(int):Item[]
parseItem(Cursor):Item
update(int,String,String):void
updateOnlineKey(int,String):void
getAllNames():String[]
<<Java Class>>
ShoppingListHandler
com.eseteam9.shoppyapp
TABLE_NAME: String
KEY_ID: String
KEY_TITLE: String
KEY_ONLINEKEY: String
KEY_ARCHIVED: String
KEY_TIMESTAMP: String
ShoppingListHandler(Context)
createTable(SQLiteDatabase):void
dropTable(SQLiteDatabase):void
add(ShoppingList):void
get(int):ShoppingList
get(String):ShoppingList
delete(int):void
existsEntry(String):boolean
existsKey(String):boolean
getAll():ShoppingList[]
update(int,String):void
updateOnlineKey(int,String):void
getCount():int
parseShoppingList(Cursor):ShoppingList
<<Java Class>>
OnlineDatabaseHandler
com.eseteam9.shoppyapp
context: Context
notifications: Notification[]
OnlineDatabaseHandler(Context)
putUser(User):void
getUser(String):void
putList(ShoppingList,User):void
getList(String):void
shareList(String,String):void
getSharedLists():void
syncList(ShoppingList):void
putItems(String,int):void
getListItems(String,int):void
checkItem(String,boolean):void
updateItem(String,Item):void
deleteItem(String):void
parseItem(Item):ParseObject
notify(int,User):void
getNotifications(String):Notification[]
<<Java Class>>
DisplayItemsActivity
com.eseteam9.shoppyapp
listId: int
listname: String
adapter: ItemAdapter
items: Item[]
DisplayItemsActivity()
onCreate(Bundle):void
onCreateOptionsMenu(Menu):boolean
onCreateContextMenu(ContextMenu,View,ContextMenuInfo):void
onContextItemSelected(MenuItem):boolean
updateView():void
checkItem(View):void
onBackPressed():void
onOptionsItemSelected(MenuItem):boolean
addDialog():void
editDialog(Item):void

The next example shows how the DisplayItemsActivity gets the Items to display. To get this data the activity creates a new ItemHandler and calls the getListItems(...) function. The ItemHandler queries the SQLite database and returns the result.
The following outline of a sequence diagram visualizes this:



This is a short and direct way to put data locally on the device, but also, as mentioned above, not compliant with the MVC-structure. Without going too much into the pros and cons of the MVC-structure and not repeating the criticism above one suggestion is in place here. In the average use of the App the direct access of the database does not seem to be a performance issue. But in regard to the OnlineDatabaseHandler the situation appears to be different. The web requests in this class take sometime anyway. This leads to an uncomfortable user experience, i.e. while adding a new item to the List a Toast “List downloaded” appears and after a confused press on cancel the List you wanted to add to looks different than before. This problem could be solved with adding a new Class to organize the communication between the model (created from the database) and the view. This would have the benefit, that the database or web requests would be made at clearly defined points in the application making the behaviour of the app more logic for the user.
Seen as a client for a web based shopping list service the app may benefit from the simple design but an added level of complexity seems to be in place to prevent wired behaviour.
But besides this point the non-orthogonal persistence strategy seems to be the right way for a shopping list app. Also the handler classes seem be be well equipped proxies to the database which make good use of the tools provided by android.

3. Activity Review

For reviewing a specific Activity we use “DisplayItemsActivty”. The Activity contains 232 lines of code in about ten methods and 4 instance variables. The responsibilities of the Activity are stated in the documentation at the top. And are: Display all Items of a list, make them clickable and show their options if pressed before. Also you should be able to add more Items to the list. This activity is like the name says responsible for displaying Items, which is a well focused thing as the Android Reference on Activity says. So the responsibility of the class is clear and not to much for a single Activity.

The variables are well named and in the context absolutely understandable what their use is.
Only mistake is “listname” which violates the convention of naming.

The onCreate() method is structured and easy to overview, but instead of just grouping them and add a comment what this group does it would be better if you just put a group into its own method and with well chosen names you could even spare the comments respectively the documentation.

The addDialog() and editDialog(Item item) methods don’t seem to fit in with the others. Why are they protected (they don’t have a visibility attribute) and not private? The only use of this two methods is in the same class, so you should make them private. Also they are very long and therefore unclear. The method doesn’t even fit on my laptop screen. There are different approaches to refactor them. One always is to create helper methods. Or you could make your own listener classes. The negativeButton needs in both methods the same onClick() method so you can put this in one CancelButtonListener, for the positiveButton you would need two separate listeners but you would spare a lot of code in the Activity which could be handled by the listener classes. In our opinion it’s not really the responsibility of the Activity to define the onclick method in it. Also the onClick method is not only there to handle the click on, but is also there to create the dialogue interface. So the responsibilities of the methods are not really clear, so they have to do more than they’re asked to do. You should delegate such things to other methods.

As said in the Overall Review you create a lot of new Instances. In this specific Activity always when you use the ItemHandler class you create a new Object which is garbage collected just after the method you directly call of it has finished because you never assign it to a variable. You do this 9 times and in some methods you do it more than once. So you could just make it an Instance variable instead of always creating and destroying a new Instance of it which is expensive and not really pretty.

The methods are also not documented. It’s not that it’s really used for all of them but instead of document it you have a normal comment on top of it, which is not totally useless but impractical. The advantage of documentation in Java is that you can use the method anywhere you want or

can and still can see what this method is used for. With your little comments you need every time you want to know what a parameter is for, to go back to the code and really look at the method. Especially for such methods as checkItem(View view) which you define in the item_row.xml as onClick() method, this also strange that this method is in the Activity because the View item_row.xml is set in the ItemAdapter.class. It is very important to say this to improve readability of the code and prevent the reader from searching and testing where this method is used and why it needs to be public. Because as a reader you think it should be private, so you make it private and see that the compiler tells you it's not used anywhere but it actually breaks the application.