Universität Bern
Introduction to Software Engineering

ShopNote

Software Review Document
v1.0
Last modified on 19.11.2013

Customer
Team 8

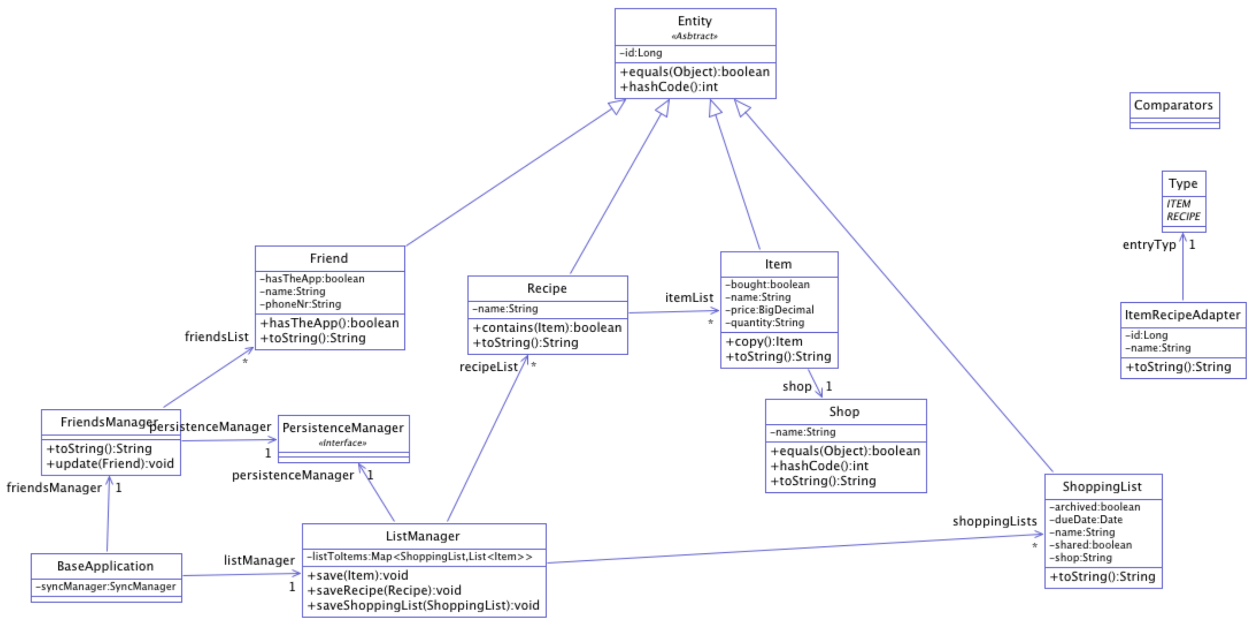
Project
Shopping-List

Autors
Raul Bolaños, Nicolas Kessler, Theodor Truffer, Lukas Zahnd

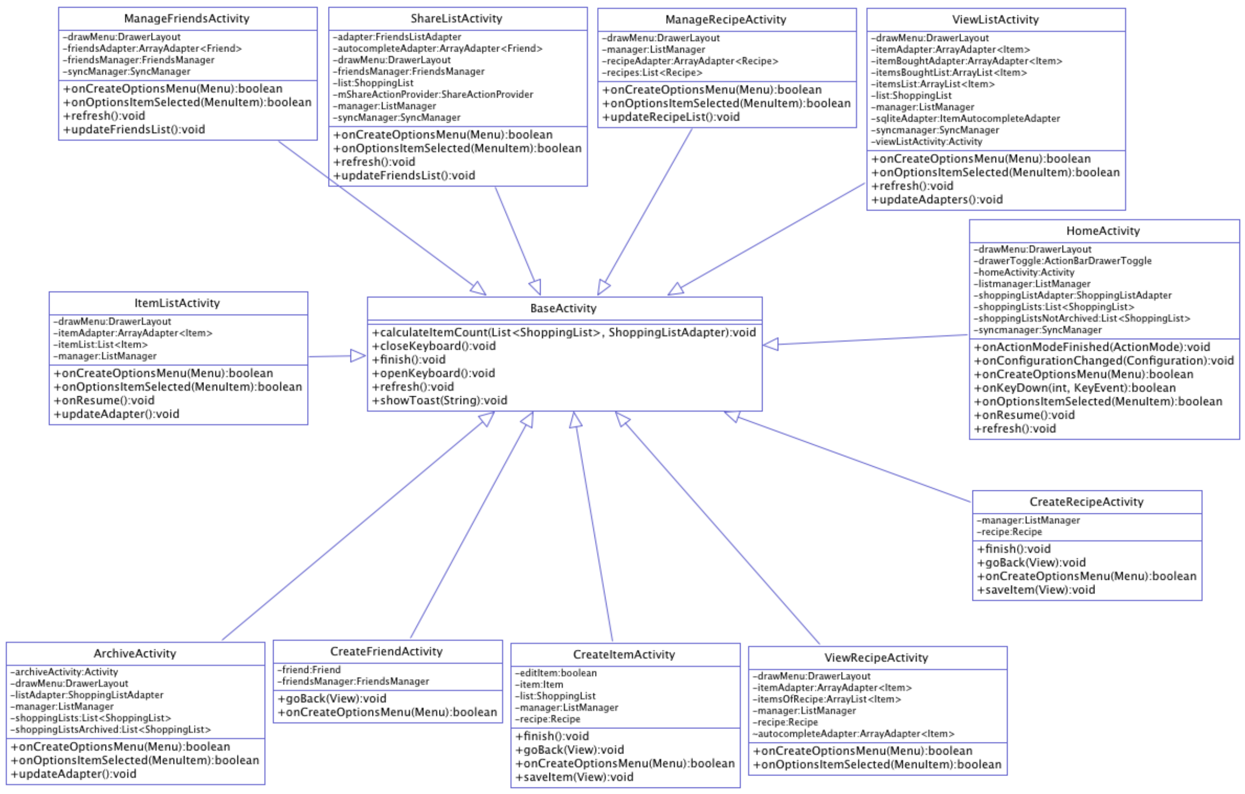
1. Review

Design

As Android itself doesn’t enforce an MVC Approach, this has to be implemented by the developers themselves. In ShopNote, the distinction between Model, View and Controller is pretty clear. The View Layer is represented by the package “Activities”, where you have all the application’s activities, which create the whole GUI. The package “Core” contains all the Objects responsible for modelling and controlling. In “Core” we have the Controller layer, that is, Friendsmanager, Persistancemanager and Listmanager.
The other objects like Item, Shop, Shoppinglist and so on are in the Model Layer. The responsibilities in these layers are defined very well. The activities don’t interact with the Model, they all have instances of the managers, which are controlling all the work with the Model, like creating Items, updating and receiving Information etc. The Activities just call the manager’s methods and uses the received Information to create the GUI or they send Information from the GUI-Input to the managers.



There are also Helper objects between view and model in the application, namely the adapters. The adapter package contains six different adapters, that works separatly, to fill the gaps between the GUI and the Model. They are well implemented and used properly. The responsibilities are very clear, and the application doesn’t have duplicated code, the organisation and reuse is good too. For example there is a “BaseActivity” class. The Activities inherit from this class, so that there is almost no duplicated code at all.


ManageFriendsActivity
-drawMenu:DrawerLayout
-friendsAdapter:ArrayAdapter<Friend>
-friendsManager:FriendsManager
-syncManager:SyncManager
+onCreateOptionsMenu(Menu):boolean
+onOptionsItemSelected(MenuItem):boolean
+refresh():void
+updateFriendsList():void
ShareListActivity
-adapter:FriendsListAdapter
-autocompleteAdapter:ArrayAdapter<Friend>
-drawMenu:DrawerLayout
-friendsManager:FriendsManager
-list:ShoppingList
-mShareActionProvider:ShareActionProvider
-manager:ListManager
-syncManager:SyncManager
+onCreateOptionsMenu(Menu):boolean
+onOptionsItemSelected(MenuItem):boolean
+refresh():void
+updateFriendsList():void
ManageRecipeActivity
-drawMenu:DrawerLayout
-manager:ListManager
-recipeAdapter:ArrayAdapter<Recipe>
-recipes:List<Recipe>
+onCreateOptionsMenu(Menu):boolean
+onOptionsItemSelected(MenuItem):boolean
+updateRecipeList():void
ViewListActivity
-drawMenu:DrawerLayout
-itemAdapter:ArrayAdapter<Item>
-itemBoughtAdapter:ArrayAdapter<Item>
-itemsBoughtList:ArrayList<Item>
-itemsList:ArrayList<Item>
-list:ShoppingList
-manager:ListManager
-sqliteAdapter:ItemAutocompleteAdapter
-syncmanager:SyncManager
-viewListActivity:Activity
+onCreateOptionsMenu(Menu):boolean
+onOptionsItemSelected(MenuItem):boolean
+refresh():void
+updateAdapters():void
HomeActivity
-drawMenu:DrawerLayout
-drawerToggle:ActionBarDrawerToggle
-homeActivity:Activity
-listmanager:ListManager
-shoppingListAdapter:ShoppingListAdapter
-shoppingLists:List<ShoppingList>
-shoppingListsNotArchived:List<ShoppingList>
-syncmanager:SyncManager
+onActionModeFinished(ActionMode):void
+onConfigurationChanged(Configuration):void
+onCreateOptionsMenu(Menu):boolean
+onKeyDown(int, KeyEvent):boolean
+onOptionsItemSelected(MenuItem):boolean
+onResume():void
+refresh():void
ItemListActivity
-drawMenu:DrawerLayout
-itemAdapter:ArrayAdapter<Item>
-itemList:List<Item>
-manager:ListManager
+onCreateOptionsMenu(Menu):boolean
+onOptionsItemSelected(MenuItem):boolean
+onResume():void
+updateAdapter():void
BaseActivity
+calculateItemCount(List<ShoppingList>, ShoppingListAdapter):void
+closeKeyboard():void
+finish():void
+openKeyboard():void
+refresh():void
+showToast(String):void
CreateRecipeActivity
-manager:ListManager
-recipe:Recipe
+finish():void
+goBack(View):void
+onCreateOptionsMenu(Menu):boolean
+saveItem(View):void
ArchiveActivity
-archiveActivity:Activity
-drawMenu:DrawerLayout
-listAdapter:ShoppingListAdapter
-manager:ListManager
-shoppingLists:List<ShoppingList>
-shoppingListsArchived:List<ShoppingList>
+onCreateOptionsMenu(Menu):boolean
+onOptionsItemSelected(MenuItem):boolean
+updateAdapter():void
CreateFriendActivity
-friend:Friend
-friendsManager:FriendsManager
+goBack(View):void
+onCreateOptionsMenu(Menu):boolean
CreateItemActivity
-editItem:boolean
-item:Item
-list:ShoppingList
-manager:ListManager
-recipe:Recipe
+finish():void
+goBack(View):void
+onCreateOptionsMenu(Menu):boolean
+saveItem(View):void
ViewRecipeActivity
-drawMenu:DrawerLayout
-itemAdapter:ArrayAdapter<Item>
-itemsOfRecipe:ArrayList<Item>
-manager:ListManager
-recipe:Recipe
~autocompleteAdapter:ArrayAdapter<Item>
+onCreateOptionsMenu(Menu):boolean
+onOptionsItemSelected(MenuItem):boolean

The only thing you could criticize about the Design of ShopNote, are the missing invariants. In the whole application there are only very few invariants to check pre- and postconditions.

The code is organized in packages which is nice but the packages could be organized in a more intuitive way and given better names. For example the package activites holds a lot of activites with very different purposes and they are basically organized by the type of class (e.g. Activity) they represent and not by their function (e.g. views, model, controller, utils, libs etc.)

Over all, the whole Design is very nice, everything is very clear, it doesn't take long to understand the interaction between the objects and it doesn't violate the principles of object oriented design.

Coding Style

Consistent style, clear names, no duplicated code. Again, there's not much to complain about. While reading the code, you can always imagine what a method-call does, even if you didn't read the method's code, because the names describe the functionality perfectly. Abstract classes and parent classes like Request, Entity or BaseActivity avoid duplicated code. Furthermore, encapsulation is kept very well. There are no unnecessary public declared methods and all fields are private and are received through getters and setters. The methods are kept short and are easy to read. The application is also handling exceptions e.g. the RequestSender catches all possible Exceptions and prints

out the respective error-message.
Again there are almost no assertions and class-invariants, so that there are few contracts to check pre- and postconditions. But except for that, the code style is very nice and clear.

Documentation

The Documentations of the classes in ShopNote explain often well the functionality of the particular class. They are not too long and not too short and sometimes even contain code snippets of how to use this class properly.
There are nevertheless some small details like in the “ListChangeRequest” class where should be specified that it is an abstract class and in “PersistenceManager” where is also not mentioned that it is an interface or in the “Request” class where the first line doesn’t really gives the most important information about this class.

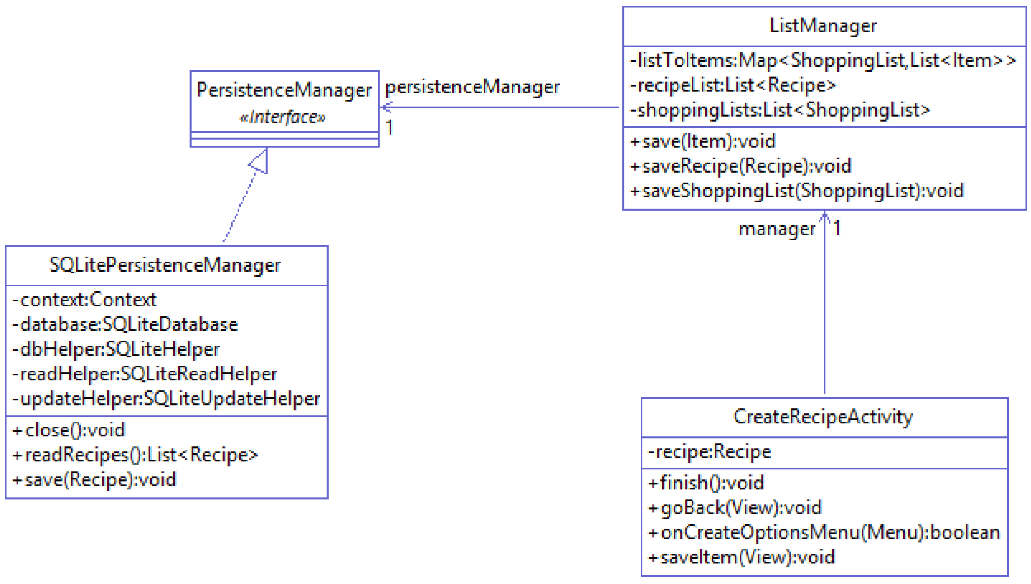
The Documentation of the methods has some small deficits. For example there is a javadoc documentation which says “everything for lists”, follow by some methods handling the lists. This is on one side unnecessary, because it is obvious that getList, save(list), and remove(list) has something to do with the lists, and on the other side the documentation is sticked to the first method, because it’s javadoc, which could be a bit confusing.
Some documentations and comments are a bit unnecessary, like comments inside a method which say “//add clicklistener” and the following line is something with .setOnItemClickListener(). But since there are only a few of these, it’s not really annoying while reading the code. And more documentation is always better than no documentation. Sometimes there are also unnecessary HTML tags inside the documentation, this could be stripped to clean up the documentation parts.
As said before, these are small deficits and beside them the documentation is very good and helpful.

Tests

The implemented Test-Cases are easy to understand. There are no comments, but they’re not needed because the test’s names are self-explaining. Everything is clear and easy to read through. The three managers are tested pretty good, there are a lot of test-methods in each of them, so the manager’s methods are covered well. But the coverage for the whole project is a bit low, since there are only these three tests and a HomeActivityTest. There are many activities, that are not tested and neither are the packages share/request.
In short, the existing tests are good, but there could be more.

2. Data Persistence Strategy

The data persistence of ShopNote is handled centrally by the PersistenceManager interface, which is implemented by the SQLitePersistanceManager.



This UML-Diagramm briefly describes the way data is saved in ShopNote, for the special case of saving a created Recipe in the CreateRecipeActivity. As you can see easily, the CreateRecipeActivity contains a ListManager object, which again contains a PersistanceManager, respectively a SQLitePersistenceManager. When the user creates a recipe in the CreateRecipeActivity, the activity calls it's saveItem() method. This method calls (amongst other things) the saveRecipe() method of it's ListManager, with the recipe to be saved as parameter. The ListManager then calls the save() method of the SQLitePersistenceManager, again with the Recipe as parameter. The SQLitePersistenceManager now saves the Recipe in the SQLiteDatabase. To do that it has a few HelperClasses. In this case, the updateHelper converts the recipe into values and the readHelper checks if the recipe is already in the database.
This is only the way recipes are saved in the database, but since the ListManager and the PersistenceManager have overloaded save() methods for every object to be saved, it's a very similar procedure with other objects.

I would really like to criticize this strategy and suggest ways to make it better, but to me this is a good way to persist data and it's well designed and implemented, so I actually can't think of details how to improve it.

3. The ItemListActivity

The ItemListActivity is the Activity that appears when you choose “Item List” in the Navigation Drawer. It’s responsible to show all the items saved in the database.
When you look at the code of ItemListActivity, you see that what it does is to get all Items from the manager, puts them in an adapter and shows them in a list. And that’s exactly what it is responsible to: It doesn’t ask the database for data, it asks the manager to get it; It doesn’t adapt the data for the view, it asks the adapter to do it.
The rest of the code has also to do with the GUI: Activity methods (onCreate, onPause, onResume etc.), Setting ClickListeners, set and show the NavigationDrawer and the “+” button in the right top corner, that leads to the CreateItemActivity. Since the ItemListActivity is in the View Layer, these methods are all in it’s responsibility. There is no logic in this Activity that could be moved to another class in the project. It does what it has to do and pushes anything else to others. And that’s ideal.