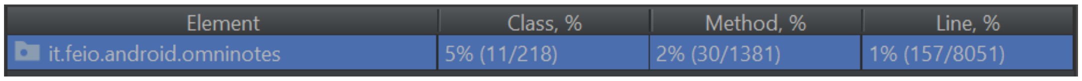
Interesting Test Cases in Omni-Notes

Jing Chen, Guowei Li, Dongxin Xiang

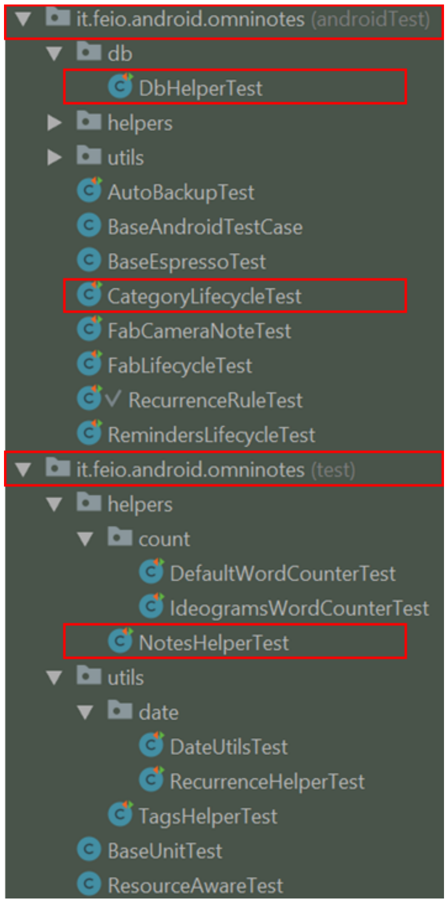
1 Introduction

According to the analysis of testing coverage, only 5% of the classes and 2% of the methods were tested in Omni-Notes (Fig 1). So there are limited test cases in the project. However, the test cases are organized well and correspond to the components such as class names and method names in the project. There are two folders containing test cases, which are AndroidTest and Test (Fig. 2). AndroidTest mainly contains functional testing and many of them are UI testing. The Test folder contains test cases related to logic in the app. Therefore, the test cases in this project are mainly divided into unit tests and UI tests.



Element	Class, %	Method, %	Line, %
it.feio.android.omninotes	5% (11/218)	2% (30/1381)	1% (157/8051)

Fig. 1 Test coverage of Omni-Notes

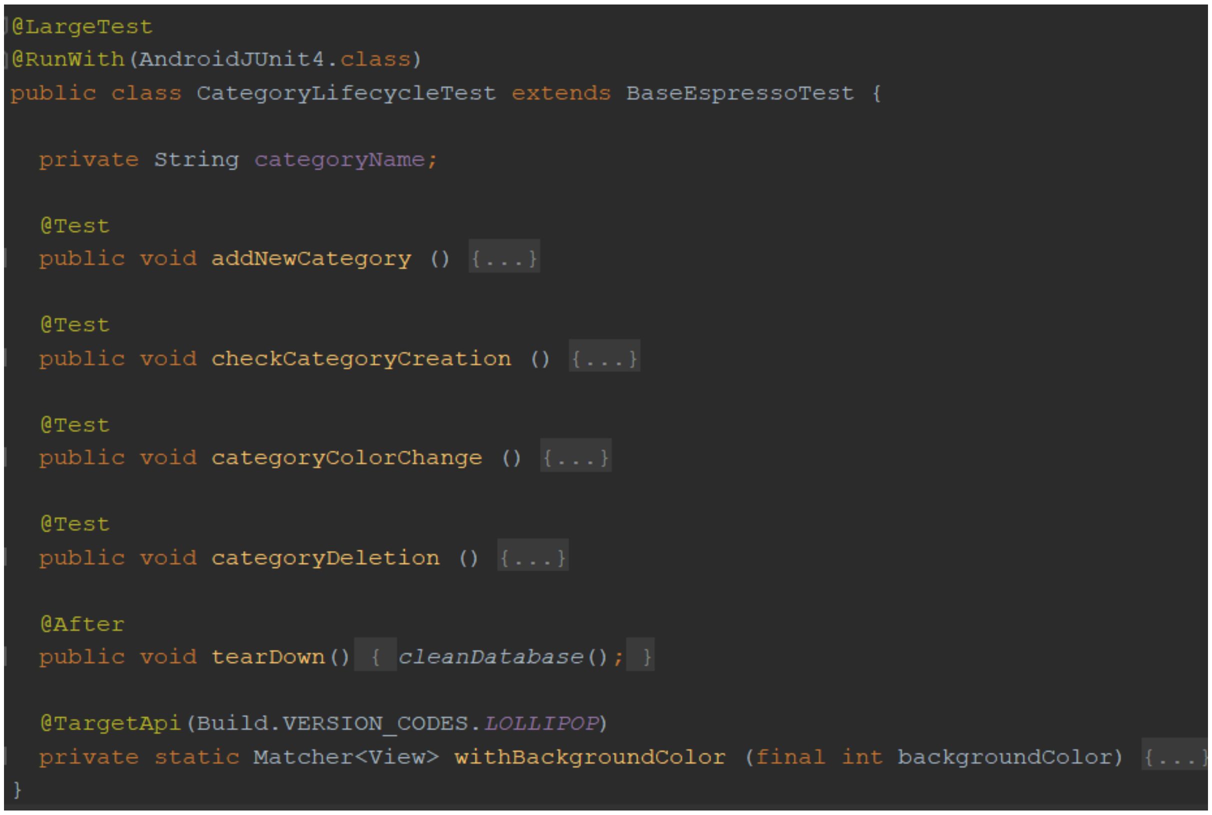


```
▼ it.feio.android.omninotes (androidTest)
    ▼ db
        DbHelperTest
    ▶ helpers
    ▶ utils
      AutoBackupTest
      BaseAndroidTestCase
      BaseEspressoTest
      CategoryLifecycleTest
      FabCameraNoteTest
      FabLifecycleTest
      ✓ RecurrenceRuleTest
      RemindersLifecycleTest
▼ it.feio.android.omninotes (test)
    ▼ helpers
        ▼ count
            DefaultWordCounterTest
            IdeogramsWordCounterTest
          NotesHelperTest
    ▼ utils
        ▼ date
            DateUtilsTest
            RecurrenceHelperTest
          TagsHelperTest
      BaseUnitTest
      ResourceAwareTest
```

Fig. 2 Test suites in Omni-Notes

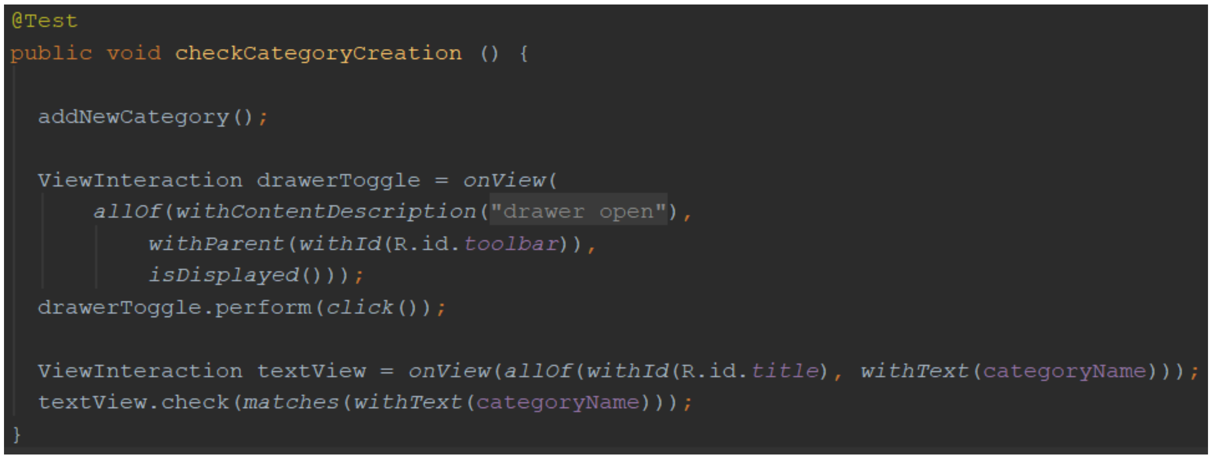
2 Interesting test cases

2.1 CategoryLifecycleTest



```
@LargeTest
@RunWith(AndroidJUnit4.class)
public class CategoryLifecycleTest extends BaseEspressoTest {

  private String categoryName;

  @Test
  public void addNewCategory () {...}

  @Test
  public void checkCategoryCreation () {...}

  @Test
  public void categoryColorChange () {...}

  @Test
  public void categoryDeletion () {...}

  @After
  public void tearDown() { cleanDatabase(); }

  @TargetApi(Build.VERSION_CODES.LOLLIPOP)
  private static Matcher<View> withBackgroundColor (final int backgroundColor) {...}
}
```

Fig. 3 Test cases for Category-related operations



```
@Test
public void checkCategoryCreation () {

  addNewCategory();

  ViewInteraction drawerToggle = onView(
      allOf(withContentDescription("drawer open"),
          withParent(withId(R.id.toolbar)),
          isDisplayed()));
  drawerToggle.perform(click());

  ViewInteraction textView = onView(allOf(withId(R.id.title), withText(categoryName)));
  textView.check(matches(withText(categoryName)));
}
```

Fig. 4 Source code of checkCategoryCreation

2.1.1 Interesting properties of the test cases

UI test cases were built in the project by the Espresso testing framework which is provided by Android. CategoryLifecycleTest is an example of the UI test where UI test cases were recorded and then customized by the testers. In CategoryLifecycleTest, the test cases (Fig. 3) are about the lifecycle of a category including creating a category, color-changing of a category, and deleting a category. The @After tearDown() method in JUnit helps clean the database after all the testing operations. @TargetApi sets the test environment including the android version. These rules make writing and organizing test cases more efficient. In Fig. 4, it shows the original code of checkCategoryCreation, which is used to check if the newly created category is displayed in the category list in the drawable toggle on the left of the screen of the app.

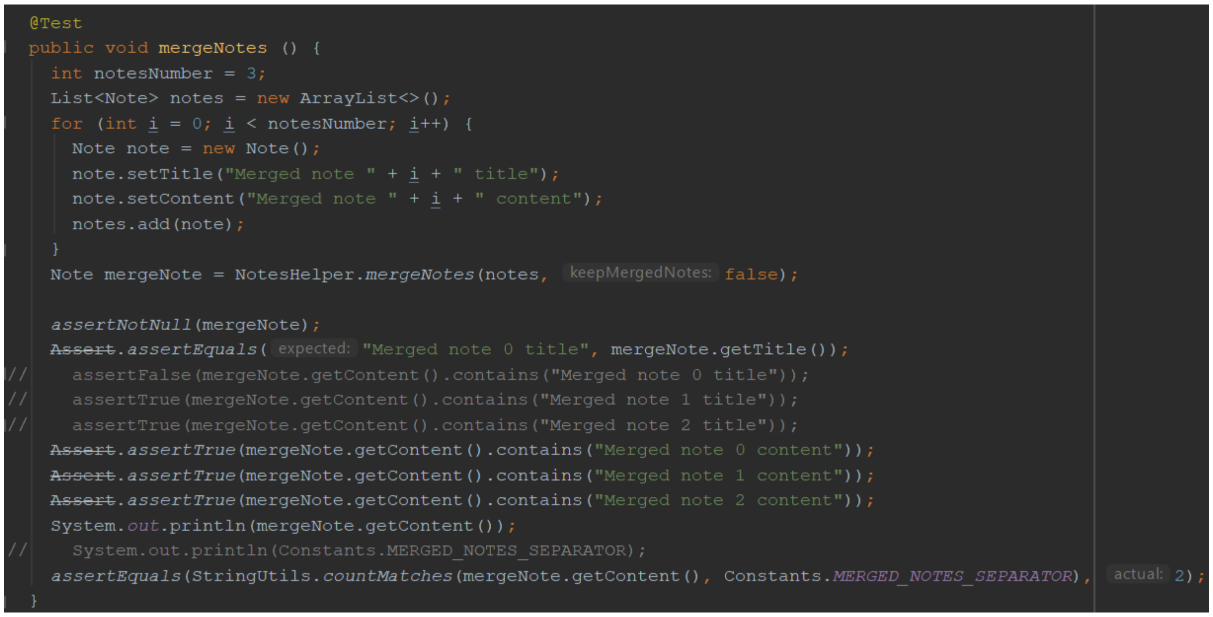
2.1.2 New understanding of the app

We found some detailed rules for creating and deleting a category. For example, when we delete a category, all the notes in this category will be marked with no specific category. And it will show a notification about this on the screen. We also found the categories can be customized in hundreds of colors, which is convenient. In our experience of using other note-taking apps, there are only around 20 colors to choose from. So it is easier to make us confused about different categories marked by the same color. Obviously, Omni-Notes is more powerful than many other similar apps.

2.2 NotesHelperTest/mergeNotes

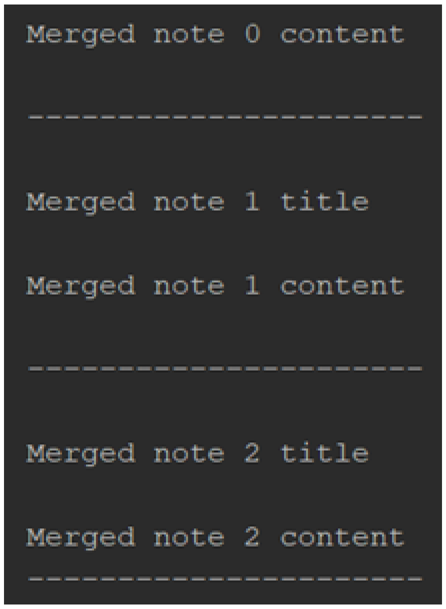
2.2.1 Interesting properties of the test cases

Note-taking and note-managing are two most important features of the app. This test case is used to test the mergeNotes method in the project (Fig. 5). Specifically, it checks if the title of the merged note is the first original note's title and if the content of the merged note contains the titles of other original notes. We also added some print statements to help understand the test cases which are those commented statements in Fig. 5 and the printed result is shown in Fig. 6. By doing this, we found the test case also tests the format of the merged note by checking if the merged note contains the MERGED_NOTES_SEPARATOR (the dashed line below in Fig. 6).



```
@Test
public void mergeNotes () {
  int notesNumber = 3;
  List<Note> notes = new ArrayList<>();
  for (int i = 0; i < notesNumber; i++) {
    Note note = new Note();
    note.setTitle("Merged note " + i + " title");
    note.setContent("Merged note " + i + " content");
    notes.add(note);
  }
  Note mergeNote = NotesHelper.mergeNotes(notes,  keepMergedNotes: false);

  assertNotNull(mergeNote);
  ~~Assert~~.assertEquals( expected: "Merged note 0 title", mergeNote.getTitle());
//    assertFalse(mergeNote.getContent().contains("Merged note 0 title"));
//    assertTrue(mergeNote.getContent().contains("Merged note 1 title"));
//    assertTrue(mergeNote.getContent().contains("Merged note 2 title"));
  ~~Assert~~.assertTrue(mergeNote.getContent().contains("Merged note 0 content"));
  ~~Assert~~.assertTrue(mergeNote.getContent().contains("Merged note 1 content"));
  ~~Assert~~.assertTrue(mergeNote.getContent().contains("Merged note 2 content"));
  System.out.println(mergeNote.getContent());
//    System.out.println(Constants.MERGED_NOTES_SEPARATOR);
  assertEquals(StringUtils.countMatches(mergeNote.getContent(), Constants.MERGED_NOTES_SEPARATOR),  actual: 2);
}
```

Fig. 5 Source code of mergeNotes



```
Merged note 0 content

----------------------

Merged note 1 title

Merged note 1 content

----------------------

Merged note 2 title

Merged note 2 content
----------------------
```

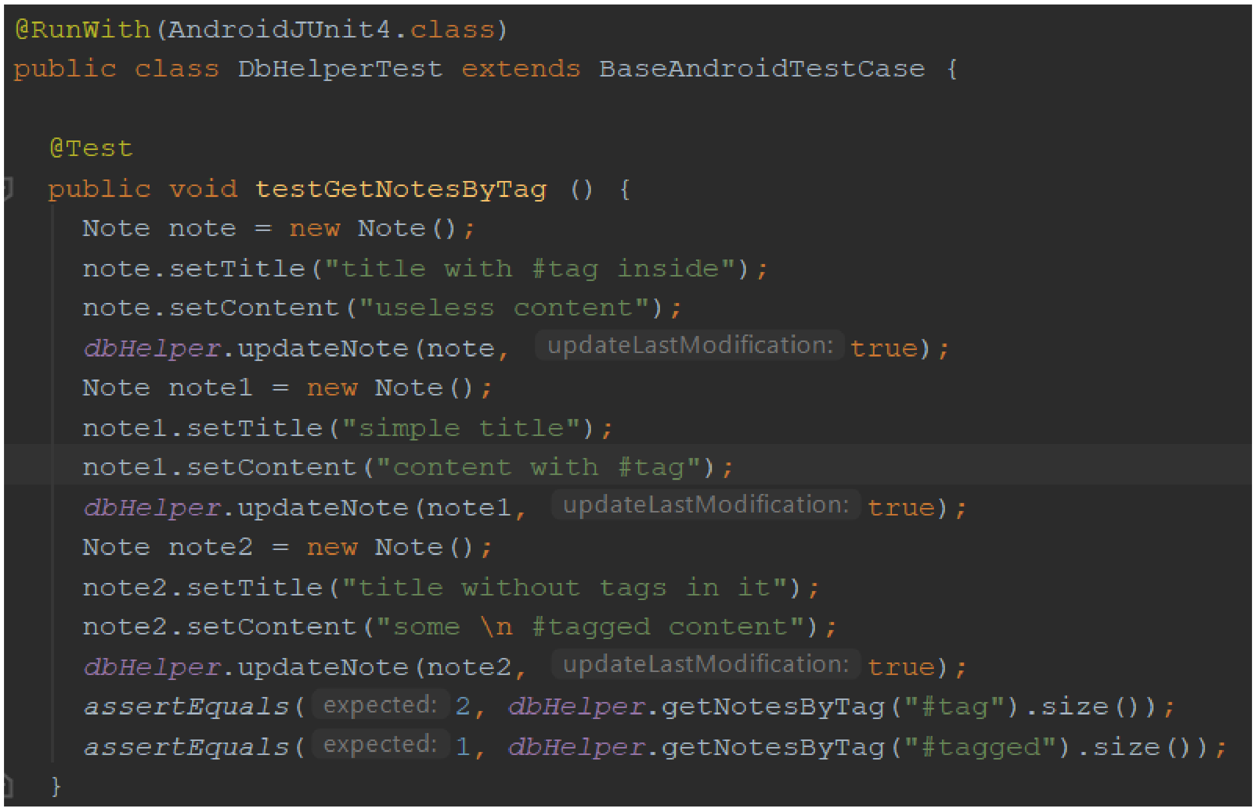
Fig. 6 Content of a merged note

2.2.2 New understanding of the app

First, we realized that Omni-Note is flexible for managing existing notes. We did not use the app merging functionality before. After finding the test case, we tried this operation on the app. Second, from the test case, we found that the app can merge more than two notes. In addition, we learned about the rules of note merging. The first note’s title will be set as the title of the merged note. And other notes’ titles will be put in the content of the merged note. What’s more, when we looked at the mergeNotes method that is tested, there are many rules for merging notes. There is a lot of other information in the original notes such as location (longitude, latitude), reminder and category. But how to merge such information is not tested. So far the tester only tested if the titles are merged correctly. In our opinion, the developer or the tester may consider correct title merging is the most important for merging operations.

2.3 DbHelperTest/testGetNotesByTag

2.3.1 Interesting properties of the test cases



```
@RunWith(AndroidJUnit4.class)
public class DbHelperTest extends BaseAndroidTestCase {

  @Test
  public void testGetNotesByTag () {
    Note note = new Note();
    note.setTitle("title with #tag inside");
    note.setContent("useless content");
    dbHelper.updateNote(note,  updateLastModification: true);
    Note note1 = new Note();
    note1.setTitle("simple title");
    note1.setContent("content with #tag");
    dbHelper.updateNote(note1,  updateLastModification: true);
    Note note2 = new Note();
    note2.setTitle("title without tags in it");
    note2.setContent("some \n #tagged content");
    dbHelper.updateNote(note2,  updateLastModification: true);
    assertEquals( expected: 2, dbHelper.getNotesByTag("#tag").size());
    assertEquals( expected: 1, dbHelper.getNotesByTag("#tagged").size());
  }
```

Fig. 7 Source code of testGetNotesByTag

Database class plays a significant role in the app since we need all the notes persistent. DbHelper is a bridge between the database and the logic of the app, to help get data from the database more conveniently. One of the useful methods in DbHelper is to get notes by a specific tag or multiple tags (getNotesByTag method). Searching by tags is a key feature of the app which is put on the home page by the developer on the website of the app. The test case (Fig. 7), testGetNotesByTag, is used to test if the getNotesByTag method can be called to get all the notes with the specified tag. In the test case, the developer just tests the size of the returned notes instead of checking if the returned notes indeed contain the specified tag. Maybe the indirect way is faster to test, but in our opinion, checking the content of the returned notes is more reliable.

2.3.2 New understanding of the app

From the test case, we found multiple tags can be used to sieve notes at the same time. Through this test case, we also dug further into the DbHelper class and obtained more information about it. The updateNote method invoked in the test case (Fig. 7) is used both to store a new note and to update an existing note to the database. And the SQLite Database used in the app is provided by Android.