UCI University of California, Irvine

Design Patterns for Omni-Notes

Guowei Li, Dongxin Xiang, Jing Chen

1. Five Design Patterns

In software engineering, a software design pattern is a general, reusable solution to a commonly occurring problem within a given context in software design. It is a description or template for how to solve a problem that can be used in many different situations.[1] In this section, we are going to state five design patterns of various categories that we identified from Omni-Notes.

1.1 Adapter

- **Category**: Structural
- **Example in Omni-Notes:**

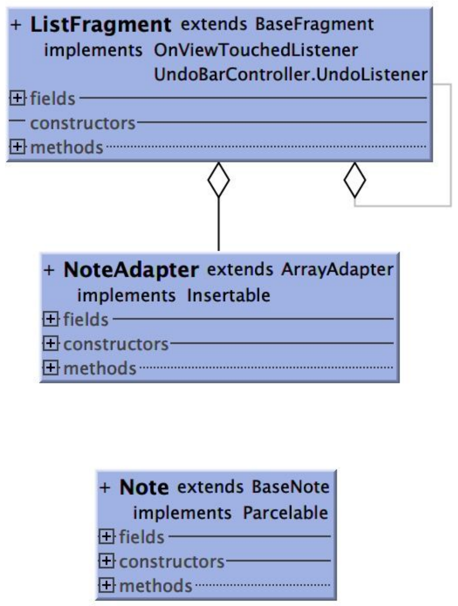


Fig.1-1-1 An Example of the Adapter Pattern in Omni-Notes

A typical example of the adapter pattern in Omni-Notes is shown in Fig.1-1-1. In this example, *NoteAdapter* extends *ArrayAdapter<Note>* and plays a role as *Note*'s adapter. It has a List<Note> type member variable named **notes** to load all existing notes created by the user. It implements methods for filling note data from a *Note* object in **notes** in the listview of *ListFragment* and updating UI components with updated data.

- **Benefits in This Example:**
  NoteAdapter works as the bridge between UI components in *ListFragment* and actual *Note* objects. It holds the data from *Note* objects and sends the data to views in *ListFragment*, which makes it possible for UI components to display shortcuts of existing notes.

1.2 Asynchronous Method Invocation

- **Category**: Concurrency

- **Example in Omni-Notes:**

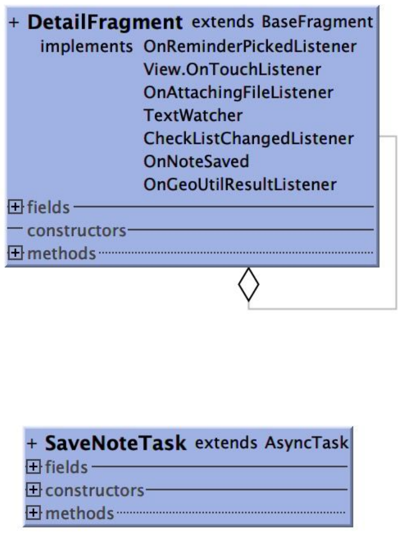
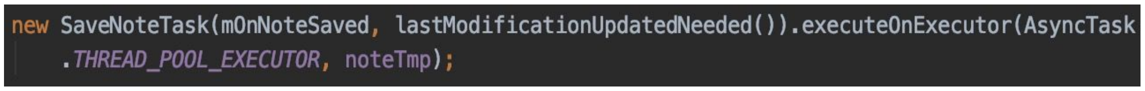


Fig.1-2-1 An Example of the Asynchronous Method Invocation Pattern in Omni-Notes



```
new SaveNoteTask(mOnNoteSaved, lastModificationUpdatedNeeded()).executeOnExecutor(AsyncTask
    .THREAD_POOL_EXECUTOR, noteTmp);
```

Figure 1-2-2 A SaveNoteTask Instance Created in saveNote() in DetailFragment

A typical example of the async method invocation pattern in Omni-Notes is shown in Fig.1-2-1. In this example, *SaveNoteTask* extends *AsyncTask* which was intended to enable proper and easy use of the UI thread. *SaveNoteTask* overrides the **doInBackground()** method to save a specific note in the background asynchronously. *DetailFragment* will create an instance of *SaveNoteTask* when a note is about to be saved due to the user's operation.

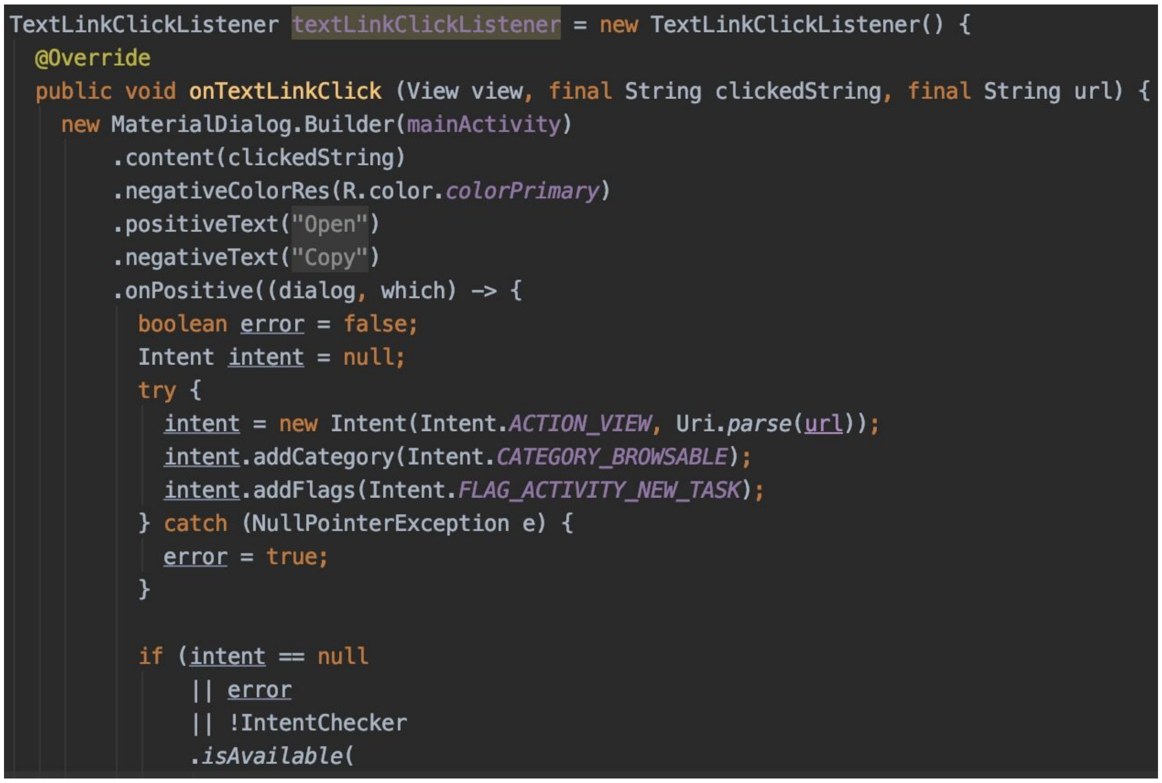
- **Benefits in This Example:**

- ❏ Multiple note-saving tasks are allowed to run in parallel. Under the extreme circumstances, the device runs slowly and the user edits multiple notes fast. But other note-saving tasks do not need to wait when a note-saving task is running.
- ❏ User operations in *DetailFragment* will not be blocked when it calls *SaveNoteTask*, which improves the performance of UI components.

1.3 Builder

- **Category**: Creational

- **Example in Omni-Notes:**



```
TextLinkClickListener textLinkClickListener = new TextLinkClickListener() {
  @Override
  public void onTextLinkClick (View view, final String clickedString, final String url) {
    new MaterialDialog.Builder(mainActivity)
        .content(clickedString)
        .negativeColorRes(R.color.colorPrimary)
        .positiveText("Open")
        .negativeText("Copy")
        .onPositive((dialog, which) -> {
          boolean error = false;
          Intent intent = null;
          try {
            intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
            intent.addCategory(Intent.CATEGORY_BROWSABLE);
            intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
          } catch (NullPointerException e) {
            error = true;
          }

          if (intent == null
              || error
              || !IntentChecker
              .isAvailable(
```

Fig. 1-3-1 An Example of the Builder Pattern in Omni-Notes

A typical example of a builder pattern in Omni-Notes is shown in Fig.1-3-1. In the overridden **onTextLinkClick()** method of a *TextLinkClickListener* instance named **textLinkClickListener**, a *MaterialDialog* object is created by using *MaterialDialog.Builder*. *MaterialDialog.java* is already implemented as a java library file and it has an inner class *Builder* which provides lots of builder methods to build a *MaterialDialog* object. When the user clicks a hyperlink in a note, a material dialog will pop up and provide two options: open or copy the link.

- **Benefits in This Example:**
  - ❏ The builder pattern provides clear separation between the construction and representation of a *MaterialDialog* object. A material dialog is made up of various views, so there are many parameters to set, which makes it complicated to build a valid *MaterialDialog* object. By applying builder pattern, we can ignore the order of parameters to be set and focus on the creation of the object.
  - ❏ The builder pattern improves the flexibility of building a *MaterialDialog* object. Some parameters are optional. By applying builder pattern, there is no need to pass in null for optional parameters to the constructor. As a result, it also improves the readability of code.

1.4 Factory

- **Category**: Creational

- **Example in Omni-Notes:**

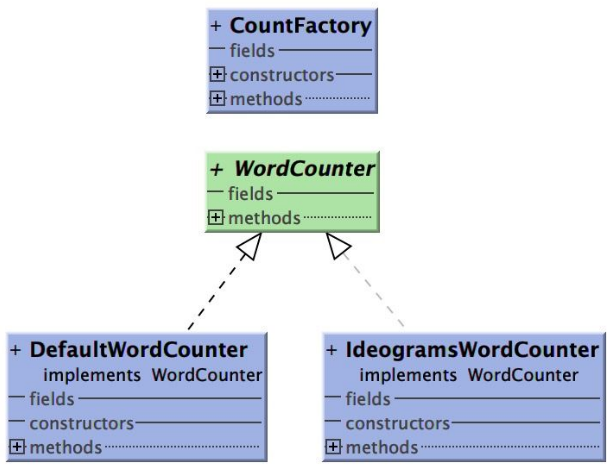
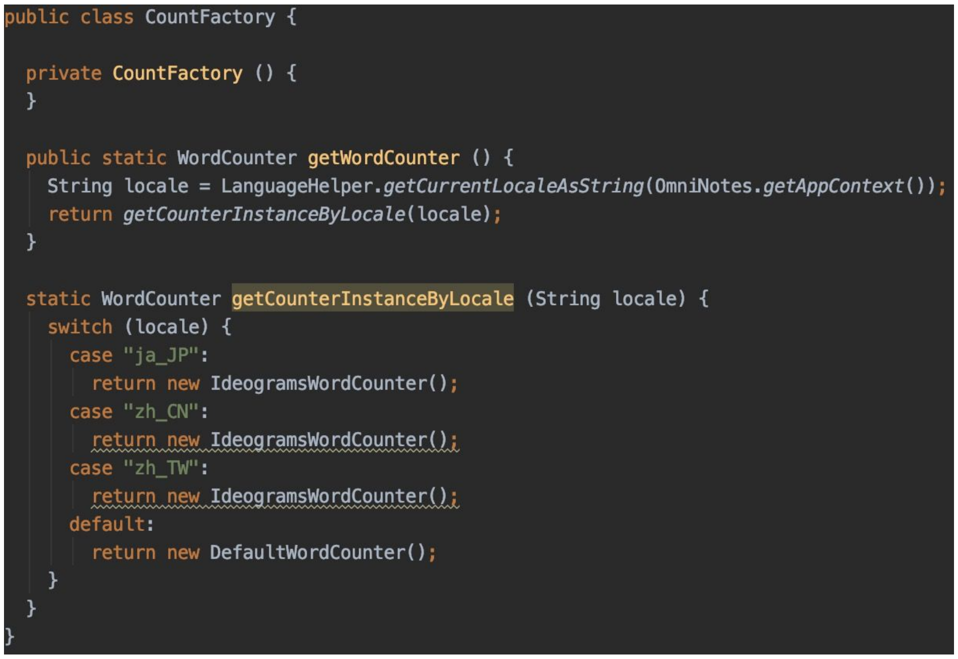


Fig.1-4-1 An Example of the Factory Pattern in Omni-Notes



```java
public class CountFactory {

  private CountFactory () {
  }

  public static WordCounter getWordCounter () {
    String locale = LanguageHelper.getCurrentLocaleAsString(OmniNotes.getAppContext());
    return getCounterInstanceByLocale(locale);
  }

  static WordCounter getCounterInstanceByLocale (String locale) {
    switch (locale) {
      case "ja_JP":
        return new IdeogramsWordCounter();
      case "zh_CN":
        return new IdeogramsWordCounter();
      case "zh_TW":
        return new IdeogramsWordCounter();
      default:
        return new DefaultWordCounter();
    }
  }
}
```

Fig. 1-4-2 Implementation of CountFactory

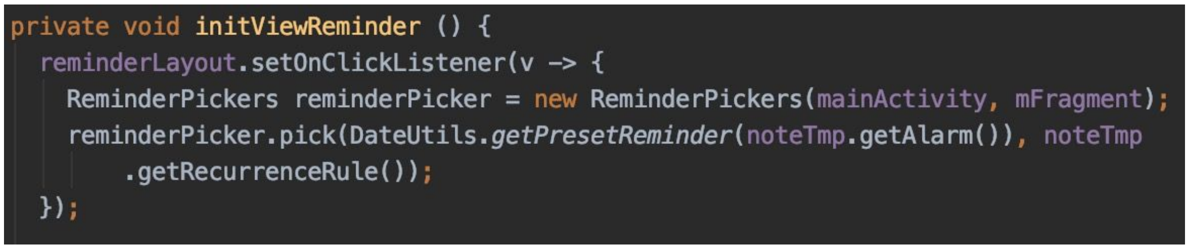
A typical example of the factory pattern in Omni-Notes is shown in Fig.1-4-1. In this example, *CountFactory* is a factory class that can produce a class implementing the interface *WordCounter* to count words in a note properly. Its **getCounterInstanceByLocale()** method will base on locale, the input string, to return a proper word counter for different languages. If the locale indicates that the language is Japanese, Simplified Chinese or Traditional Chinese, it will return an *IdeogramsWordCounter* object. In other cases, it will return a *DefaultWordCounter* object.

- **Benefits in This Example:**
  - ❏ The factory pattern lets the factory decide output type based on input when it cannot anticipate the class of objects it must create.
  - ❏ The factory pattern makes code more robust, less coupled and easy to extend. For example, in the *CountFactory*, if developers want to add a new type of word counter, they can simply implement the interface *WordCounter* and add a new case to the **getCounterInstanceByLocale()** method without modifying massive blocks of code.

1.5 Callback

- **Category**: Idiom

- **Example in Omni-Notes:**



```
private void initViewReminder () {
  reminderLayout.setOnClickListener(v -> {
    ReminderPickers reminderPicker = new ReminderPickers(mainActivity, mFragment);
    reminderPicker.pick(DateUtils.getPresetReminder(noteTmp.getAlarm()), noteTmp
        .getRecurrenceRule());
  });
```

Fig. 1-5-1 An Example of the Callback Pattern in Omni-Notes

A typical example of the callback pattern in Omni-Notes is shown in Fig.1-5-1. In this example, this code block in the *onClickListener* (in **initViewReminder()** in *DetailFragment.java*) is passed as an argument of **setOnClickListener()** and works as a callback. When the method **initViewReminder()** gets called and runs, the inner code block will not be executed immediately. Only when the reminder layout is clicked, it will get executed so that a reminder picker will be shown on the screen for the user to set time for a reminder.

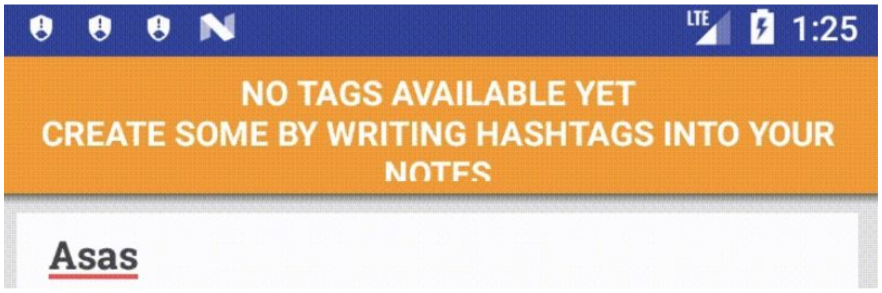
- **Benefits in This Example:**
  - ❏ The callback pattern is extremely useful in UI implements, since UI components are supposed to respond only when users take corresponding operations.
  - ❏ By using the callback pattern, we can avoid blocking the main thread when the app is running.

2. Contribution for Solving the First Issue

In this part, we will state our contribution for solving the first issue we chose step by step. FIrst, we would like to briefly describe the chosen issue. Then we will introduce the issue-solving process, including the issue reproduction, our thinking process and final solution to the issue. Finally we put screenshots of the pull request to make a visually straightforward presentation of our contribution.

2.1 Issue Description

- Url: https://github.com/federicoiosue/Omni-Notes/issues/663
- Context:
  - Device: Galaxy Nexus
  - OS Version: Android 7.0
  - API Version: 24
- Description:
  Text cut in the last line in no tags available message. Also, it is difficult to read all of the text in the message within the duration as it is very short.


1:25

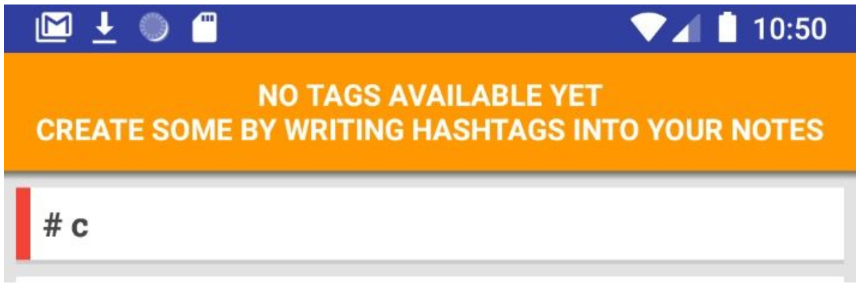
NO TAGS AVAILABLE YET
CREATE SOME BY WRITING HASHTAGS INTO YOUR
NOTES

Asas

Fig. 2-1-1 Text Cut in No Tags Available Message

2.2 Issue Solution

- Emulator Context:
  - Device: Pixel 2
  - OS Version: Android 8.1
  - API Version: 27
- Issue Reproduction:
  First, we created some notes without tags in Omni-Notes. Then, we chose one of the notes, clicked it and entered the note editing interface. When clicking the tag icon on the top-right of the screen, no tags available message showed up. In our issue reproduction, the message duration was indeed very short, as described above. However, there was no text cut situation, even if we used several types of emulators (Fig. 2-2-1).

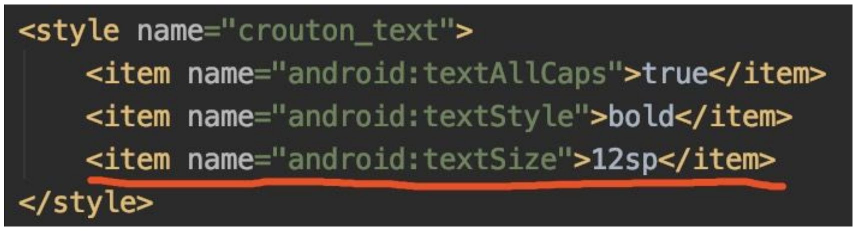

10:50

NO TAGS AVAILABLE YET
CREATE SOME BY WRITING HASHTAGS INTO YOUR NOTES

c

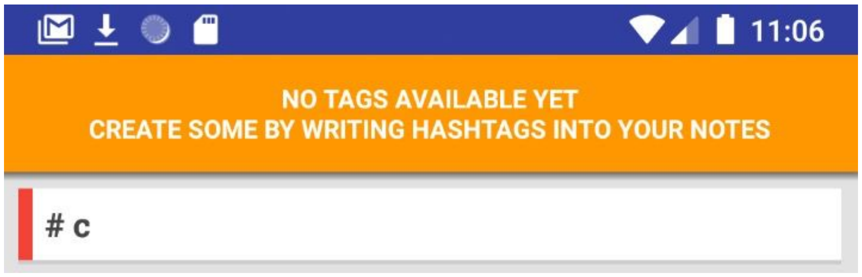
Fig. 2-2-1 No Text Cut in No Tags Available Message

- Exploration and Solution:
  1) Solution for Text Cut
  Based on the issue description and our issue reproduction, we made an assumption that text cut situations may only appear on some kinds of devices. Since the default text size is 14sp, we adjusted it to 12sp by adding an item of text size to the style for Crouton text in *style.xml* (Fig. 2-2-2). However, the adjusted text size seemed to be too small for message text to be clearly read (Fig. 2-2-3). Therefore, we decided not to change the text size since there were no text cut problems on many types of devices and text with the bigger size is easier to read.



```xml
<style name="crouton_text">
    <item name="android:textAllCaps">true</item>
    <item name="android:textStyle">bold</item>
    <item name="android:textSize">12sp</item>
</style>
```

Fig. 2-2-2 Added Item of Text Size

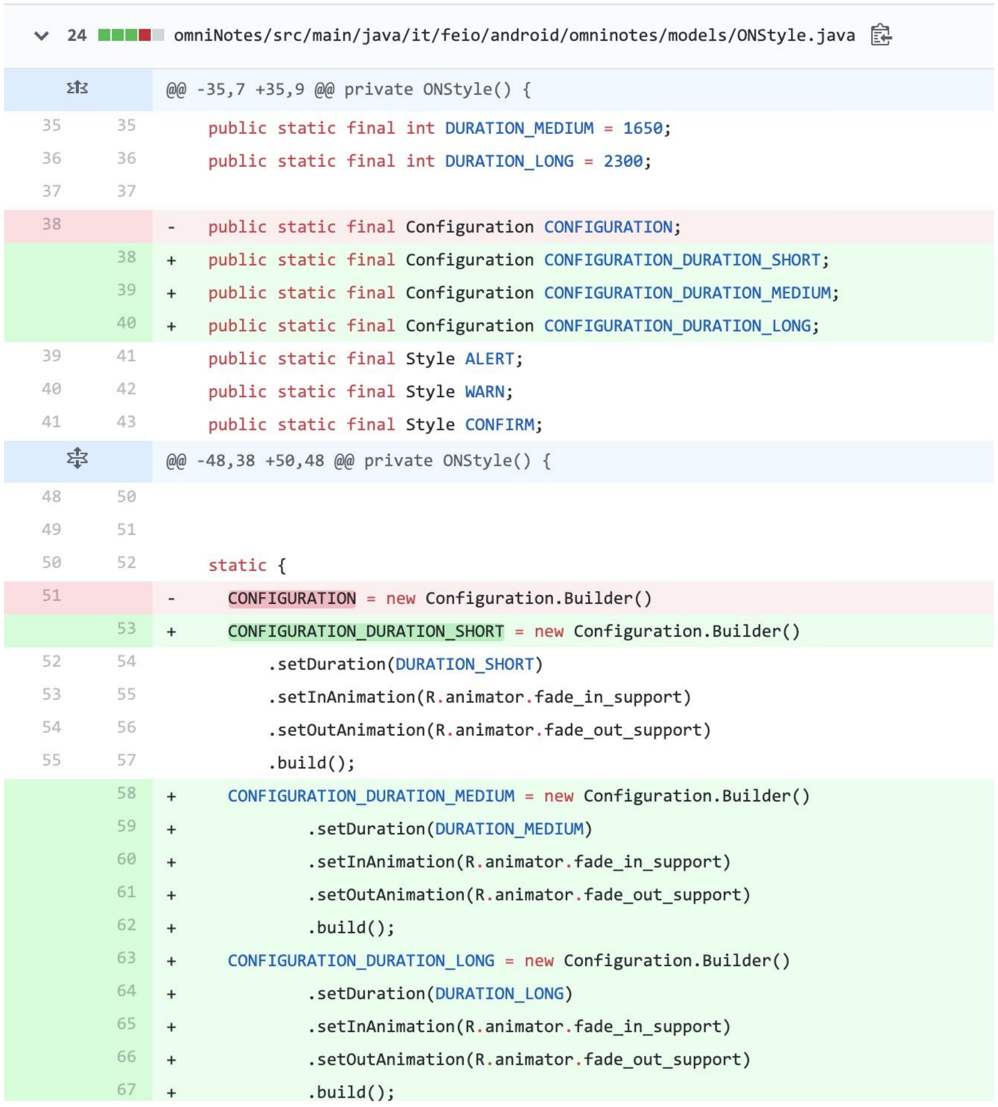


NO TAGS AVAILABLE YET
CREATE SOME BY WRITING HASHTAGS INTO YOUR NOTES

Fig. 2-2-3 Display of No Tags Available Message After Modification

2) Solution for Short Message Duration

The first thing we did was to find where the message was located. We achieved that by searching the text in the project. We found the string is stored in a string item named **no_tags_created** in *strings.xml* and it is used as one of the two arguments for **showMessage()** called in **addTags()** in *DetailFragment*. According to the code in **addTags()**, we found that **showMessage()** is called only when the user wants to add tags for note(s) and there are no existing tags. The other argument is **ONStyle.WARN**, which is a public member value of Style type. Then by looking into *OnStyle.java,* we found that there was one *Configuration* object called **CONFIGURATION** in it as a member value. It was the only one configuration for all styles created in *ONStyle* and set the duration to DURATION_SHORT which represents a short time length of 950ms. We modified the configuration's name to **CONFIGURATION_DURATION_SHORT** and added two other configurations named **CONFIGURATION_DURATION_MEDIUM** and **CONFIGURATION_DURATION_LONG**. The former defines a medium time length of 1650ms and the latter defines a long time length of 2300ms. We applied **CONFIGURATION_DURATION_MEDIUM** on **ONStyle.WARN**. After modification, the no tags available message stays longer on the screen, thus we fixed issue #663.

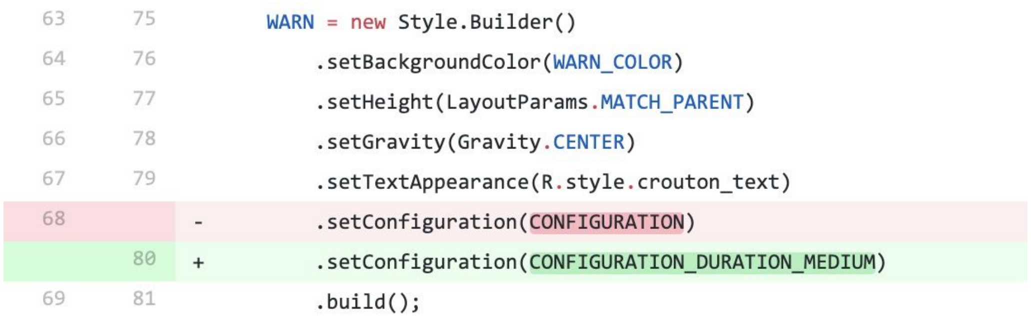


```
24 omniNotes/src/main/java/it/feio/android/omninotes/models/ONStyle.java
@@ -35,7 +35,9 @@ private ONStyle() {
    public static final int DURATION_MEDIUM = 1650;
    public static final int DURATION_LONG = 2300;

-   public static final Configuration CONFIGURATION;
+   public static final Configuration CONFIGURATION_DURATION_SHORT;
+   public static final Configuration CONFIGURATION_DURATION_MEDIUM;
+   public static final Configuration CONFIGURATION_DURATION_LONG;
    public static final Style ALERT;
    public static final Style WARN;
    public static final Style CONFIRM;
@@ -48,38 +50,48 @@ private ONStyle() {


    static {
-     CONFIGURATION = new Configuration.Builder()
+     CONFIGURATION_DURATION_SHORT = new Configuration.Builder()
          .setDuration(DURATION_SHORT)
          .setInAnimation(R.animator.fade_in_support)
          .setOutAnimation(R.animator.fade_out_support)
          .build();
+     CONFIGURATION_DURATION_MEDIUM = new Configuration.Builder()
+             .setDuration(DURATION_MEDIUM)
+             .setInAnimation(R.animator.fade_in_support)
+             .setOutAnimation(R.animator.fade_out_support)
+             .build();
+     CONFIGURATION_DURATION_LONG = new Configuration.Builder()
+             .setDuration(DURATION_LONG)
+             .setInAnimation(R.animator.fade_in_support)
+             .setOutAnimation(R.animator.fade_out_support)
+             .build();
```

Fig. 2-2-4 Modification on ONStyle.java (1)

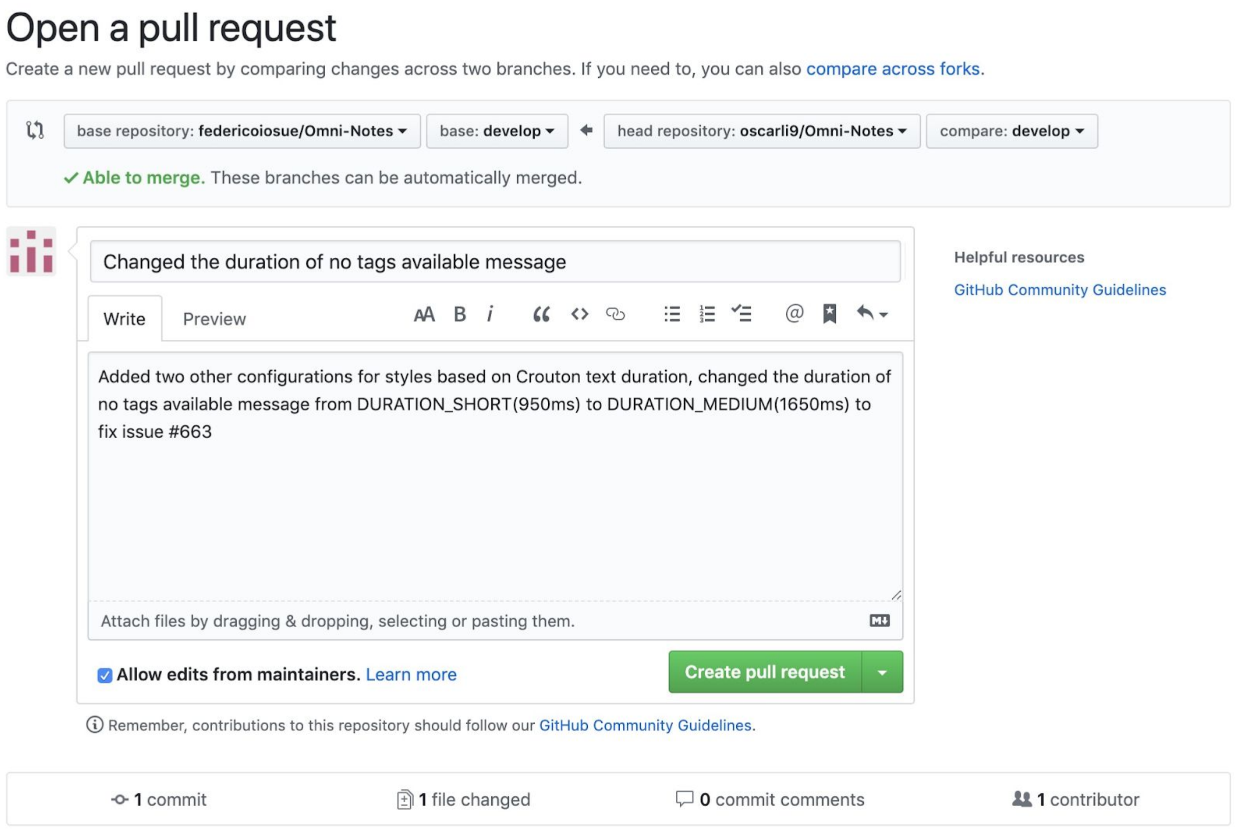


```
      WARN = new Style.Builder()
          .setBackgroundColor(WARN_COLOR)
          .setHeight(LayoutParams.MATCH_PARENT)
          .setGravity(Gravity.CENTER)
          .setTextAppearance(R.style.crouton_text)
-         .setConfiguration(CONFIGURATION)
+         .setConfiguration(CONFIGURATION_DURATION_MEDIUM)
          .build();
```

Fig. 2-2-5 Modification on ONStyle.java (2)

2.3 Pull Request

After the modification of ONStyle.java, we submitted a pull request to the Omni-Notes repo. The process is shown below in Fig. 2-3-1.


Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.

base repository: **federicoiosue/Omni-Notes** base: **develop** ← head repository: **oscarli9/Omni-Notes** compare: **develop**

✓ **Able to merge.** These branches can be automatically merged.

Changed the duration of no tags available message

Added two other configurations for styles based on Crouton text duration, changed the duration of no tags available message from DURATION_SHORT(950ms) to DURATION_MEDIUM(1650ms) to fix issue #663

Attach files by dragging & dropping, selecting or pasting them.

✓ **Allow edits from maintainers.** Learn more

Create pull request

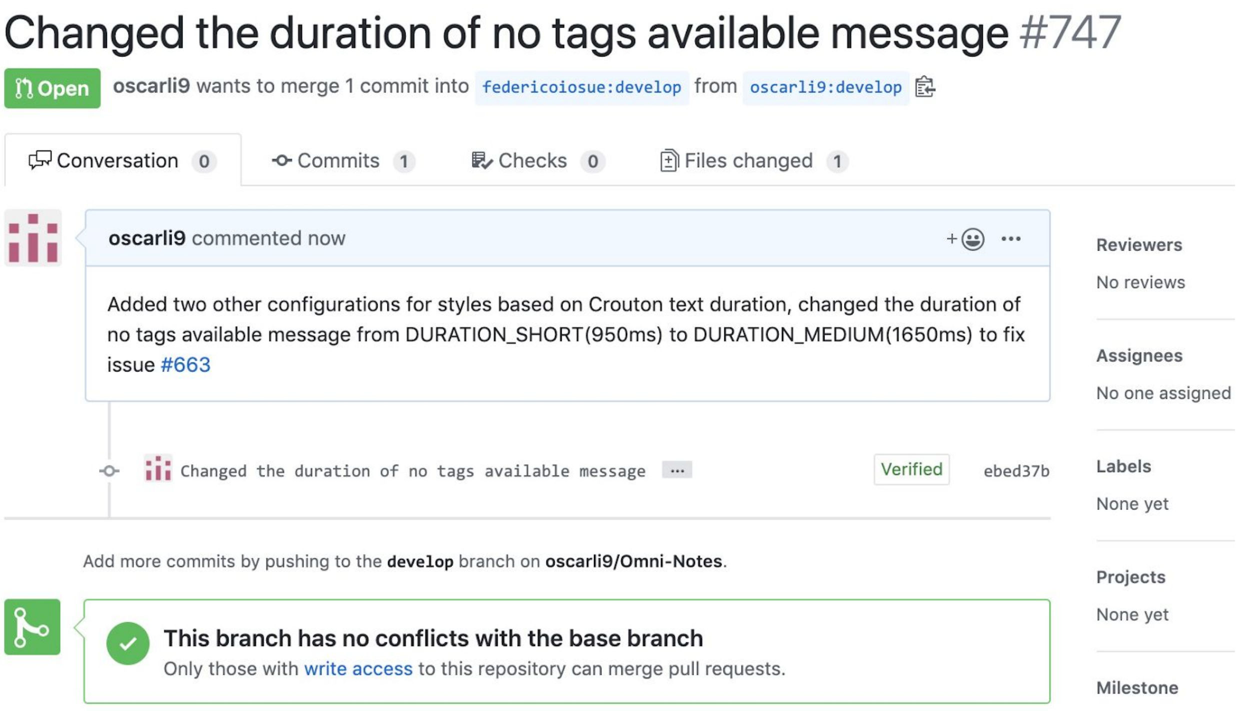
Remember, contributions to this repository should follow our GitHub Community Guidelines.

Helpful resources

GitHub Community Guidelines

1 commit **1** file changed **0** commit comments **1** contributor

Fig. 2-3-1 Pull Request for Fixing Open Issue #663



Changed the duration of no tags available message #747

Open oscarli9 wants to merge 1 commit into `federicoiosue:develop` from `oscarli9:develop`

oscarli9 commented now

Added two other configurations for styles based on Crouton text duration, changed the duration of no tags available message from DURATION_SHORT(950ms) to DURATION_MEDIUM(1650ms) to fix issue #663

`Changed the duration of no tags available message` Verified ebed37b

Add more commits by pushing to the **develop** branch on **oscarli9/Omni-Notes**.

This branch has no conflicts with the base branch
Only those with write access to this repository can merge pull requests.

Fig. 2-3-2 Pull Request After Submission

References

[1] Software design pattern in Wikipedia. https://en.wikipedia.org/wiki/Software_design_pattern