

Stanford University, CS 193A

Homework 3: Mad Libs





(Pairs allowed)

Assignment idea and spec by Marty Stepp.

This assignment practices writing apps with multiple activities, instance state and data, and input/output files.

This is a **pair assignment**. You may complete it alone or with a partner. Please read our [Pairs web page](#) for some of our policies and suggestions if you do plan to work in a pair.

Starter Files:

-  [madlibs-starter-files.zip](#) (ZIP of all starter files)
-  [Story.kt](#) (provided helper code)
-  [madlibs.png](#) (logo image)
-  [clothing.txt](#), [dance.txt](#), [marty.txt](#), [tarzan.txt](#), [university.txt](#), [wannabe.txt](#) (Mad Lib story input files)

File and Project Names:


- Your **Android Studio project**'s name must be EXACTLY the following, case-sensitive, including the underscores:

- cs193a_hw3_**SUNETID** (example: cs193a_hw3_ksmith12)

Or if you are working in a pair, please list both partners' SUNetIDs separated by an underscore:

- cs193a_hw3_**SUNETID1_SUNETID2** (example: cs193a_hw3_ksmith12_tjones5)

- You must have activity classes with the exact names given below under "Program Description," case sensitive, in corresponding file names (e.g. class FooActivity in file **FooActivity.kt**). You may add other activities, classes, and files to your app if you like, but you must have the exact activity file names below at a minimum, spelled and capitalized exactly as written here.

- When you are finished, ZIP your project ([instructions](#)) into a file with the following exact file name, including spelling and capitalization:
 - cs193a_hw3_**SUNETID**.zip (example: cs193a_hw3_ksmith12.zip)
 - or cs193a_hw3_**SUNETID1_SUNETID2**.zip if you work in a pair (example: cs193a_hw3_ksmith12_tjones5.zip)
- If you use any Android **libraries** in your app, make sure they are located in **app/libs/** and are included in your ZIP that you turn in. Please ask the instructor or a TA if you plan to use a third-party library that has not been discussed in class.
- Turn in link:  (You only need to turn in one copy if you work in a pair.)

Program Description and Functionality Requirements:

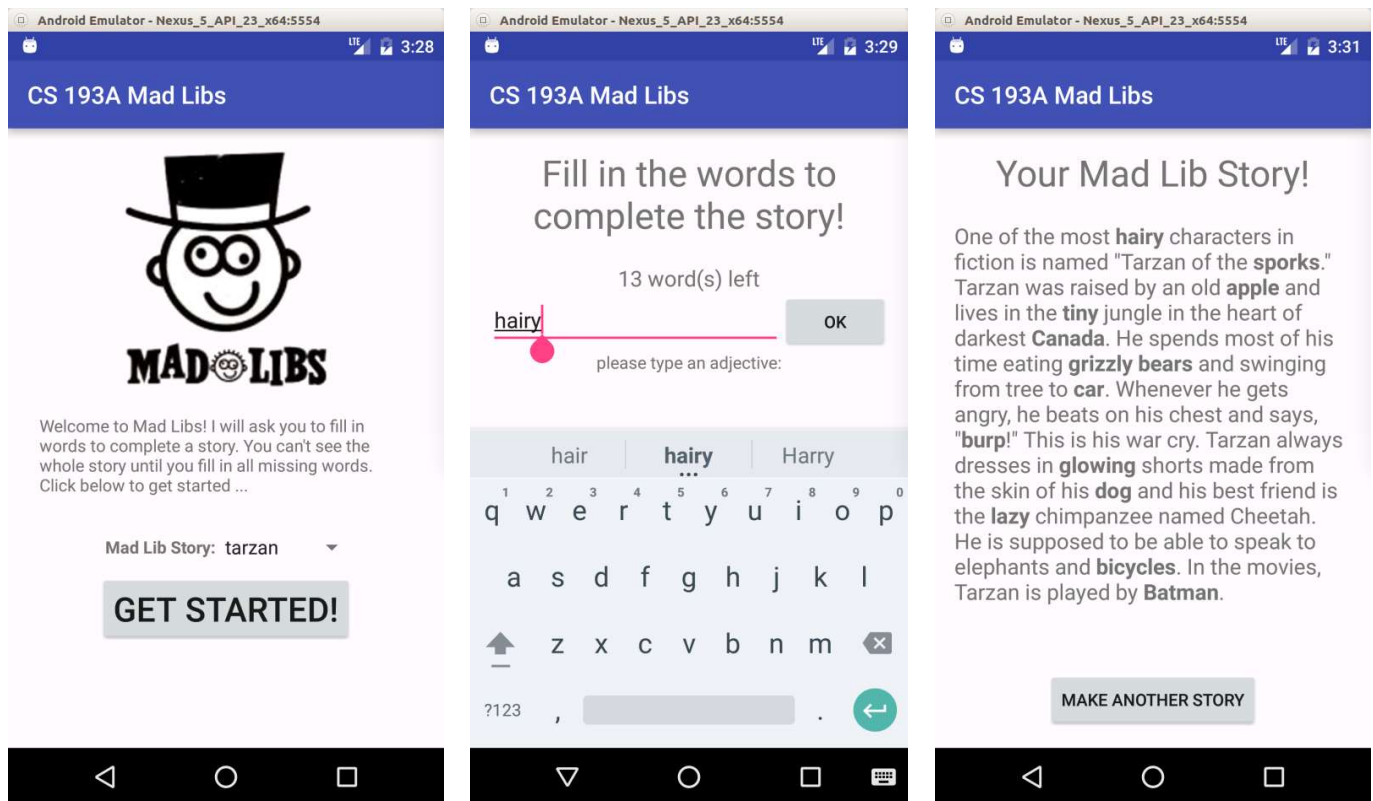
"**Mad Libs**" are short stories that have blanks called **placeholders** to be filled in. In the non-computerized version of this game, one person asks a second person to fill in each of the placeholders without the second person knowing the overall story. Once all placeholders are filled in, the second person is shown the resulting silly story.

For this assignment, write an Android app that reads in a Mad Lib from a text file in a specific format. The text file represents placeholders as tokens that start and end with < > brackets, like <adjective> or <proper noun>. Your app reads the file, looks for any such placeholders, and prompts the user to replace them with specific words. Once the user has typed in replacements for all placeholders, the completed story is shown on the screen.

Since one of the main goals of this assignment is to practice writing an app with multiple activities, we require you to have at least the following **three activities** in your app:

- **MainActivity**: An initial "welcome" screen explaining the app.
- **FillInWordsActivity**: a screen that repeatedly prompts the user to fill in placeholders.
- **ShowStoryActivity**: a third activity to display the completed story.

The screenshots below indicate the flow of the UI for your app. Of course you don't need to exactly match our sample's UI, but you are welcome to make yours match our appearance if you like.



Screenshots of app's three activities in order

The following sections describe each activity in detail. In addition, you are required to implement **one custom or extra feature** as described later in this document.

MainActivity:

Your MainActivity must have the following features:

- Shows a Mad Libs logo, such as the one provided in our starter code as [madlibs.png](#).
- Shows some text explaining the program to the user.
- A way of choosing between various mad lib stories. In our screenshots, we use a [Spinner](#), though you can use buttons, a list view, or any other reasonable choice. You don't need to list every provided .txt file from the starter code, so long as your app has at least 2-3 files that can be chosen. You can use your own story templates if you like, rather than the ones we have provided.
- A way of choosing a story template file and advancing to the next step, which is the FillInWordsActivity.

You will probably need to pass some information between the activities of your app. For example, in this MainActivity, you may need to pass to the FillInWordsActivity information about which story the user chose. Recall that you can do this putting "extra" parameters inside of your [Intent](#) objects that you use to invoke an activity.

FillInWordsActivity:

Your FillInWordsActivity must have the following features:

- A prompt that asks the user to fill in each placeholder from the story template.
- A place for the user to type their replacement text to fill in each placeholder. You must allow the user to type multi-word strings and strings that contain punctuation characters, though it is okay if your app disallows the < and > characters.
- Some kind of indication about how much progress the user has made and/or how many placeholders are left to fill in. In our screenshots, we have a text view that shows how many placeholders remain, but you could show a progress bar or anything reasonable for this.
- Once all placeholders are filled in, your code should advance to the third and final step, which is the ShowStoryActivity.

ShowStoryActivity:

Your ShowStoryActivity must have the following features:

- A display of the filled-in Mad Lib story. (If the story is too long to fit on one screen, you must provide a way to scroll it, such as using a [ScrollView](#).)
- A way to go back to the start of the app, which is the MainActivity.

Mad Lib Input Files:

We provide several Mad Lib story text files you can use. You should put them in your project's **app/src/main/res/raw/** folder, such as **tarzan.txt**.

```
One of the most <adjective> characters in fiction is named
"Tarzan of the <plural noun>." Tarzan was raised by an old
<noun> and lives in the <adjective> jungle in the
heart of darkest <place>. He spends most of his time
eating <plural noun> and swinging from tree to <noun>.
Whenever he gets angry, he beats on his chest and says,
"<funny noise>!" This is his war cry. Tarzan always dresses in
<adjective> shorts made from the skin of his <noun>
and his best friend is the <adjective> chimpanzee named
Cheetah. He is supposed to be able to speak to elephants and
<plural noun>. In the movies, Tarzan is played by <person's name>.
```

*Provided Mad Lib input file, **tarzan.txt***

The best way to break apart this text is to use a [Scanner](#), which has a next() method that breaks apart the text by whitespace and returns the next token. Or better yet, use our provided Story

class described below.

Provided Story Class:

Unlike the past two assignments, this assignment comes with optional **starter code** provided by the instructor and TAs. The code for reading the story text file, breaking it apart, looking for the placeholders, etc. is not particular to Android. To make the assignment more manageable, if you want a head start toward implementing this particular option, we provide a starter file named **Story.kt** that you can optionally use as a building block.

If you use it in your project, you can construct a `Story` object and then call its `read` method to read a story from an input file. You can pass the `read` method the name of a raw resource file as a string (such as "tarzan"), or an input stream or `Scanner`, and it will read the text data from that source, break the text apart, and find the placeholders for you, etc. The `Story` object has other methods for filling in the placeholders later. If you use this helper object, you can focus more on the Android-specific parts of this assignment and less on the string / text processing parts. Or if you don't want to use the provided `Story` class and would rather try to write the story parsing logic yourself, that is fine.

The following are the methods of a `Story` object:

Member	Type	Description
<code>val st = Story()</code>	constructor	Constructs a new empty story.
<code>st.isHtmlMode</code>	property	Whether the story uses HTML formatting when you call <code>toString</code> on it; default <code>false</code> .
<code>st.placeholders[index]</code>	property	Returns the placeholder at the given 0-based index without its surrounding <code><></code> brackets, such as "adjective". Read-only.
<code>st.placeholderCount</code>	property	Returns the total number of placeholders in the story. Read-only.
<code>st.aVsAn(s)</code>	method	Returns "an" if the given string starts with a vowel (AEIOU), or "a" if not.
<code>st.fillPlaceholder(index, text)</code>	method	Replaces the unfilled placeholder at the given 0-based index with the given text.
<code>st.read(source)</code> <code>st.read(activity, source)</code>	method	Reads the initial story text from the given input source. You can pass the source in several ways: as an <code>InputStream</code> , or a <code>Scanner</code> , or as a raw resource ID (<code>Int</code>) or raw resource name

		(String). If you pass it as a raw resource, you must also pass the activity (this) to help load the resource.
<code>st.startsWithVowel(s)</code>	method	Returns true if the given string starts with a vowel (AEIOU).
<code>st.toString()</code>	method	Returns the story's complete text as a string, with any placeholders filled in.

Optional: If you put HTML tags into your story, you can give it basic formatting. You can make the Story object emit HTML text, specifically "bold" tags around placeholder text, by setting the story's `isHtmlMode` property to true. You can put HTML content into any TextView with code such as:

```
myTextView.setText(Html.fromHtml(myStoryText))
```

As with other assignments, here are some other miscellaneous functionality requirements that you must follow in this program:

- It is fine if your app uses "toasts" to pop up some messages, but you should not put any important/critical information in toasts only.
- Your app does not need to match the exact appearance of the screenshots in this document. But you should use a reasonable **layout** so that the various widgets are visible and can be interacted with on a standard AVD phone device such as a Nexus 5X.
- Your submission must be **your own work** and not written by someone else.
- Your submission must be **new work** for CS 193A and cannot be a project you submitted for another class, such as CS 108 or CS 147.

Extra Features:

You are required to add one **custom or extra feature** to your app beyond the ones previously described. This can be any non-trivial addition you like; the goal is to encourage you to make a unique submission and demonstrate creativity. The following are several examples of extra features that would satisfy our requirement. You may add more than one extra feature if you like.

- **Talking Mad Libs:** Use text-to-speech to make the program speak aloud the prompts to the user. If you want to get fancy, you could use speech-to-text to let the user speak answers to fill in each placeholder.
- **Sound/music:** Make your program play sound effects and/or background music while in use.
- **Activity to create new story templates:** Provide another activity (and a corresponding option in the main menu) where the user can type in a new story template in the format of

the existing .txt files. This will allow the program to grow and have more stories available to fill in for the user.

- **Advanced layout with fragments:** Make the app use a different layout in landscape mode. Avoid redundancy between layouts by using fragments and/or including layouts inside each other.
- **Robust app that maintains state:** Make the app remember its place even if the screen is rotated, or the app is exited/entered, etc. Use the activity lifecycle methods taught in class.
- **Mad lib stats:** Keep track of the number of mad libs created, number of placeholders filled in, etc. Show this information to the user, either when they finish filling in a mad lib story or in a stats activity.

Style Requirements:

(These are the same as in Homeworks 1-2.)

This course will generally have more lenient coding style requirements than a class like CS 106A/B. But we do have some important style requirements below that we ask you to follow. If you do not follow all of these requirements, you will not receive full credit for the assignment.

- **Comments:** Write a **comment header** in your main activity **.kt** file containing your name and email address along with the name of your app and a very brief description of your program, along with any special instructions that the user might need to know in order to use it properly (if there are any). Also write a brief comment header at the top of every **method** in your **.kt** code that explains the method's purpose. All of these comments can be brief (1-2 lines or sentences); just explain what the method does and/or when/why it is called. The following is a reasonable example of a comment header at the top of a **.kt** activity file:

```
1 /*
2  * Kelly Smith <ksmith12@stanford.edu>
3  * CS 193A, Winter 2049 (instructor: Mrs. Krabappel)
4  * Homework Assignment 3
5  * NumberGame 2.05 - This app shows two numbers on the screen and asks
6  * the user to pick the larger number. Perfect for Berkeley students!
7  * Note: Runs best on big Android tablets because of 1000dp font choice.
8  */
```

- **Redundancy:** If you perform a significant operation that is exactly or almost exactly the same in multiple places in your code, avoid this redundancy, such as by moving that operation into a helping method. See the lecture code from our lecture #1 for examples of this.
- **Reduce unnecessary use of global variables and private fields:** While you are allowed to have private fields ("instance variables") in your program, you should try to minimize

fields unless the value of that field is used in several places in the code and is important throughout the lifetime of the app.

- **Reduce mutability:** Use `val` rather than `var` as much as possible when declaring Kotlin variables. Use immutable collections instead of mutable ones where appropriate.
 - **Naming:** Give variables, methods, classes, etc. descriptive names. For example, other than a for loop counter variable such as `int i`, do not give a variable a one-letter name. Similarly, if you give an `id` property value to a widget such as a `TextView`, apply a similar style standard as if it were a variable name and choose a descriptive name.
-

Survey: After you turn in the assignment, we would love for you to fill out our optional [anonymous CS 193A homework survey](#) to tell us how much you liked / disliked the assignment, how challenging you found it, how long it took you, etc. This information helps us improve future assignments.

Honor Code Reminder: Please remember to follow the **Honor Code** when working on this assignment. Submit your own work and do not look at others' solutions. Also please do not give out your solution and do not place a solution to this assignment on a public web site or forum. If you need help, please seek out our available resources to help you.

Copyright © Stanford University and Marty Stepp. Licensed under Creative Commons Attribution 2.5 License. All rights reserved.