

# Stanford University, CS 193A

## Homework 5: Animal Game

### (Pairs allowed)














*Assignment spec by Marty Stepp.*

*Special thanks to the web site [animalgame.com](http://animalgame.com) for their data and assignment idea.*


This assignment practices accessing data, such as from web REST APIs, SQL databases, and remote cloud databases ([Firebase](#)).

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:

-  [animalgame-starter-files.zip](#) (ZIP of all starter files)
  -  [animalgame\\_logo.png](#) (logo image)
  -  [animalgame.json](#) (JSON data for Firebase database)
  -  [google-services.json](#) (Google JSON file to allow Firebase access)
  -  [animalgame\\_small.sql](#) (SQL data for local database; smaller version)
  -  [animalgame.sql](#) (SQL data for local database; larger version)
  -  [DBTypes.kt](#) (helper classes you may want to use to represent fetched data; optional)
  -  [SettingsActivity.kt](#) (helper class to initialize your database)
  -  [activity\\_settings.xml](#) (solution to SettingsActivity layout)
  -  [AnimalGameMainActivity.kt](#) (helper class for initial activity)
  -  [activity\\_animal\\_game\\_main.xml](#) (solution to AnimalGameMainActivity layout)
  -  [PlayGameActivity.kt](#) (helper class to play games)
  -  [activity\\_play\\_game.xml](#) (solution to PlayGameActivity layout)

### File and Project Names:

- **NEW:** If you use our Firebase, your Java classes must be in exactly the following **package**. If you used a different package, you can right-click your package in the left Project panel and click Refactor → Rename. (Try closing / reopening the project in Android Studio if renaming your package makes your project stop running properly.)
  - `cs193a.stanford.edu.animalgame`
- Your **Android Studio project's** name must be EXACTLY the following, case-sensitive, including the underscores:
  - `cs193a_hw5_SUNETID` (example: `cs193a_hw5_ksmith12`)
 Or if you are working in a pair, please list both partners' SUNetIDs separated by an underscore:
  - `cs193a_hw5_SUNETID1_SUNETID2` (example: `cs193a_hw5_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.java**). 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_hw5_SUNETID.zip` (example: `cs193a_hw5_ksmith12.zip`)
  - or `cs193a_hw5_SUNETID1_SUNETID2.zip` if you work in a pair (example: `cs193a_hw5_ksmith12_tjones5.zip`)
- If you use libraries from the Maven repository (ones that are added to a project by declaring them in your **build.gradle** file), these will be auto-downloaded when we compile your project to grade it.
- Turn in link:  [Turn in HW5 here.](#)

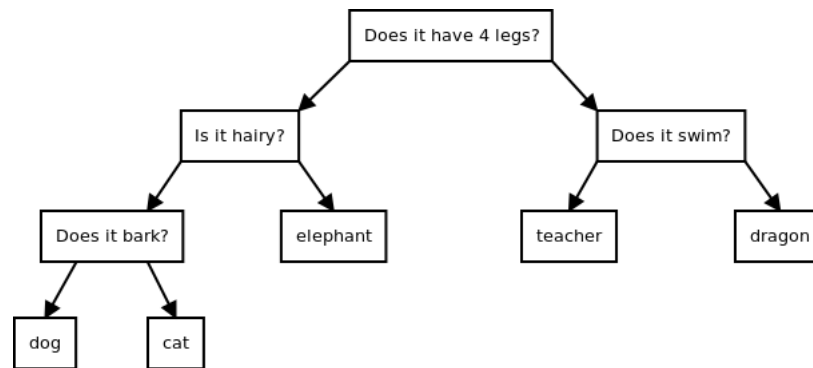
## Program Description and Functionality Requirements:

We will write a basic quiz game app named Animal Game, based on a similar game located on the web at [animalgame.com](http://animalgame.com). In this game, you, the human player, are supposed to think of any animal you like. The computer will try to guess what animal you are thinking of by asking you a series of **Yes-or-No questions**. Eventually the computer will have asked enough questions that it thinks it knows what object you are thinking of, and it will make a final guess about what your animal is. If this guess is correct, the computer wins; if not, you win.

The computer keeps track of a **database** that contains questions and answers. You could think of the data as though it were a **binary tree** where every question is a node that has two children: its Yes node and its No node. An answer node is a leaf; it has no children. The idea is that this tree can be traversed to ask the human player a series of questions. (There are several variations of this game, such as one called "20 Questions," but our game will not limit the tree's height.)

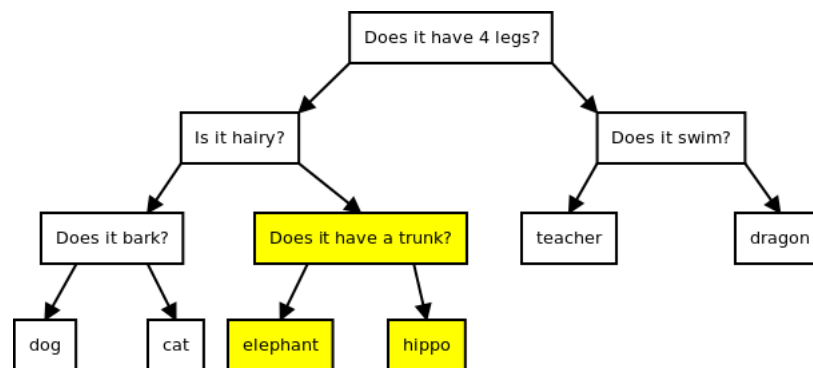
For example, in the tree below, the computer would begin the game by asking the player, "Does it have 4 legs?" If the player says "yes," the computer goes left to the "yes" subtree and then asks the user, "Is it hairy?" If the user had instead said "no," the computer would go right to the "no" subtree and then ask the user, "Does it swim?"

This pattern continues until the game reaches a leaf "answer" node. Upon reaching an answer node, the computer asks whether that answer is the correct answer (such as, "Is it a dog?"). If so, the computer wins. If not, the human wins.



computer's data decision tree (*animalgame\_small* data set)

The tree above is not very large, implying that initially the computer is not very intelligent. But the computer has the ability to **grow smarter** each time it loses a game. If the computer's answer guess is incorrect, you must give it a new question it can ask to help it in future games. For example, suppose in the preceding log that the player was thinking of a hippo. If the player answered the questions honestly, they'd say that Yes, it has 4 legs; No, it isn't hairy; and No, it isn't an elephant. At this point, the computer loses the game and asks, "What is a question I could have asked to distinguish a hippo from an elephant?" Suppose that the player supplies the question, "Does it have a trunk?" The computer could modify its decision tree to look like the figure below:



computer's data decision tree after a new question node and answer are added

We would like you to write the following activity classes:

- **AnimalGameMainActivity**: shows the app name, logo, and has buttons for the user to start playing or modify settings. When the user presses the Play button, it launches the **PlayGameActivity**.
- **PlayGameActivity**: Lets the user play the animal game. The game asks the user Yes/No questions, and the user can press the large Y and N buttons to indicate his/her answer. This will cause the game to ask a follow-up question repeatedly until the game is ready to guess the player's animal. Once the game is over, your activity must provide some way (button, etc.) for the user to play again.
- **SettingsActivity**: Has several buttons for loading and manipulating database data. This is a place where you can put commands to set up your database a single time before the game playing begins. We provide some of this code for you to help you get started setting up the initial database contents.



You do not need to exactly match the appearance of the screenshots in this document. Please feel free to be creative and customize your app as you like, so long as your app contains the functionality described below.

## Data:

The tree of animal questions and answers is stored as a **database**, represented in three formats provided by course staff. You need to pick one of the three database formats and write your app to fetch the data in that format; you do not need to use all three types of databases. Pick the format you like the best and write your application to query the data in that way. Any format is fine; each has its pros and cons, and we encourage to use whichever you like. The **Web REST API** is probably the simplest to implement, while the **Firestore** option is probably the most useful for learning about industry-level big data management.

### 1) Web REST API:

We provide a publicly readable web API for the animal game data set at the following URL:

- [http://www.martystepp.com/animalgame\\_small/api.php](http://www.martystepp.com/animalgame_small/api.php)

The data served by the REST API is very similar to the **SQL database** described on the next page, so you may also want to read that information even if you do not use SQL in your solution.

### Retrieving nodes: `get`

The API provides the following `get` action for fetching a node:

- [http://www.martystepp.com/animalgame\\_small/api.php?action=get&kind=node&nodeid=ID](http://www.martystepp.com/animalgame_small/api.php?action=get&kind=node&nodeid=ID)

Using `action=get` with `kind=node` returns a JSON object representing the row from the nodes table with the given `nodeid` value. If no node matches the given `nodeid`, an HTTP 404 error occurs and no JSON response is returned. For example, using the `animalgame_small` data shown, the query `api.php?action=get&kind=node&nodeid=1` returns the following JSON data:

```
1 {
2   "nodeid": 1,
3   "type": "question",
4   "text": "Does it have 4 legs?"
```

```

5  "userid": 0,
6  "timestamp": "0000-00-00 00:00:00",
7  "viewcount": 0
8  }

```

web API JSON response for a node

Using **action=get** with **kind=graph** allows you to search for edges from the graph table.

- [http://www.martystepp.com/animalgame\\_small/api.php?action=get&kind=graph&graphid=ID&parentid=ID&childid=ID&type=](http://www.martystepp.com/animalgame_small/api.php?action=get&kind=graph&graphid=ID&parentid=ID&childid=ID&type=)

This request will search for any graph nodes that match all of the given parameters. The graphid, parentid, childid, and type parameters are all optional; at least one of the four must be specified. If no rows match the query given, an empty array will be returned. You most likely want to use the parentid and/or childid parameters. The web query returns a JSON **array** of every graph edge that matches the constraints you gave it. For example, the URL of [api.php?action=get&kind=graph&parentid=2](http://www.martystepp.com/animalgame_small/api.php?action=get&kind=graph&parentid=2) returns the following JSON data, representing all graph nodes that have a parentid of 2:

```

1  {
2  "matches": [
3    {
4      "graphid": 103,
5      "parentid": 2,
6      "childid": 3,
7      "type": "yes"
8    },
9    {
10     "graphid": 104,
11     "parentid": 2,
12     "childid": 6,
13     "type": "no"
14   }
15 ]
16 }

```

web API JSON response for a graph search

### Adding/modifying nodes: set

The API also includes an **set** action for adding or modifying nodes.

- add a new answer node:  
[http://www.martystepp.com/animalgame\\_small/api.php?action=set&user=SUNetID&kind=node&action=set&text=text&type=an](http://www.martystepp.com/animalgame_small/api.php?action=set&user=SUNetID&kind=node&action=set&text=text&type=an)
- add a new question node:  
[http://www.martystepp.com/animalgame\\_small/api.php?action=set&user=SUNetID&kind=node&action=set&text=text&type=qu](http://www.martystepp.com/animalgame_small/api.php?action=set&user=SUNetID&kind=node&action=set&text=text&type=qu)
- modify existing node:  
[http://www.martystepp.com/animalgame\\_small/api.php?action=set&user=SUNetID&kind=node&action=set&nodeid=id&text=te](http://www.martystepp.com/animalgame_small/api.php?action=set&user=SUNetID&kind=node&action=set&nodeid=id&text=te)
- add a new graph node:  
[http://www.martystepp.com/animalgame\\_small/api.php?action=set&user=SUNetID&kind=graph&parentid=id&childid=id&type](http://www.martystepp.com/animalgame_small/api.php?action=set&user=SUNetID&kind=graph&parentid=id&childid=id&type)
- modify existing graph node:  
[http://www.martystepp.com/animalgame\\_small/api.php?action=set&user=SUNetID&kind=graph&graphid=id&parentid=id&chi](http://www.martystepp.com/animalgame_small/api.php?action=set&user=SUNetID&kind=graph&graphid=id&parentid=id&chi)
- clear ALL data added/modified by a given user (resets database to its original state):  
[http://www.martystepp.com/animalgame\\_small/api.php?action=clear&user=SUNetID](http://www.martystepp.com/animalgame_small/api.php?action=clear&user=SUNetID)

To use **action=set**, you must also pass a **user** parameter indicating your Stanford SUNetID, so that each student's modifications to the data can be kept separate. You must also pass parameters indicating the kind of node to add along with each member of that node object. For example, using the **animalgame\_small** data, the following request will add a graph node :

- [http://www.martystepp.com/animalgame\\_small/api.php?action=set&user=SUNetID&kind=graph&graphid=199&parentid=444&](http://www.martystepp.com/animalgame_small/api.php?action=set&user=SUNetID&kind=graph&graphid=199&parentid=444&)

Note that once you start modifying data with the **set** action, you'll need to pass your **user** parameter even when getting data out, so that the API knows to use your user-specific version of the data set.

- retrieve a node (user-specific):  
[http://www.martystepp.com/animalgame\\_small/api.php?action=get&user=SUNetID&kind=node&nodeid=id](http://www.martystepp.com/animalgame_small/api.php?action=get&user=SUNetID&kind=node&nodeid=id)

One common bug you may run into is that some strings do not work well when used as a REST API parameter in a URL. In particular, strings with characters such as spaces and question marks will confuse the REST API because those characters have special meaning in a URL. To fix this problem, you should *URL-encode* your text when passing it as a query parameter using the `URLEncoder` class from `java.net` package, like the code shown below:

```

1 // example of encoding a multi-word punctuated string for use as a URL query parameter
2 val example = "How are you?"
3 val encoded = URLEncoder.encode(example, "utf-8") // "How+are+you%3F"

```

For any kind of query, if you replace **animalgame\_small** with **animalgame**, the query will use the large data set rather than the small test one. Please write and debug your code using the small data set to reduce load on our server; test the large data set only after you have verified your small version thoroughly.

To help you debug, the API has an additional action called **dump** that displays all of the nodes of a given kind as an HTML table. This action is not meant to be consumed by your Android app, but rather by your web browser. To perform a data set dump, use a query such as the following:

- display all 'node' nodes as a web page:  
[http://www.martysteepp.com/animalgame\\_small/api.php?action=dump&user=SUNetID&kind=node](http://www.martysteepp.com/animalgame_small/api.php?action=dump&user=SUNetID&kind=node)
- display all 'graph' nodes as a web page:  
[http://www.martysteepp.com/animalgame\\_small/api.php?action=dump&user=SUNetID&kind=node](http://www.martysteepp.com/animalgame_small/api.php?action=dump&user=SUNetID&kind=node)
- display all of both kinds of nodes as a web page:  
[http://www.martysteepp.com/animalgame\\_small/api.php?action=dump&user=SUNetID&kind=both](http://www.martysteepp.com/animalgame_small/api.php?action=dump&user=SUNetID&kind=both)

Due to the large data size, the **dump** action can be performed only on the **animalgame\_small** data set, not on the full large **animalgame** data set.

## 2) Local SQLite database format:

We provide a series of **.sql** files above. The file **animalgame\_small.sql** declares two tables named **nodes** and **graph** that contain the following data. There is also a file **animalgame.sql** that contains a much larger data set for testing after you've finished writing the game. (Don't test on the very large data set until the small one works perfectly for several tests.)

Each row of the **nodes** table represents a question or answer node in the computer's tree. The relevant columns contain the node's ID number, its type ("question" or "answer"), and the text of the item itself. Questions are complete sentences such as "Does it bark?" while answers contain only the item itself, such as "dog". This table contains a few columns we don't care about for this assignment, such as a timestamp and view count. These are relics from the original data set acquired by the instructor to create this assignment.

**You may assume that the root of the overall question tree is a node with 'nodeid' value of 1.** Though all of the question nodes below have text that ends with a "?" character, and none of the answer nodes do, you should not assume this to be true in the general case.

nodeid	type	text	userid	timestamp	viewcount
1	question	Does it have 4 legs?	0	'0000-00-00 00:00:00'	0
2	question	Is it hairy?	0	'0000-00-00 00:00:00'	0
3	question	Does it bark?	0	'0000-00-00 00:00:00'	0
4	answer	dog	0	'0000-00-00 00:00:00'	0
5	answer	cat	0	'0000-00-00 00:00:00'	0
6	answer	elephant	0	'0000-00-00 00:00:00'	0
7	question	Does it swim?	0	'0000-00-00 00:00:00'	0
8	answer	teacher	0	'0000-00-00 00:00:00'	0
9	answer	dragon	0	'0000-00-00 00:00:00'	0

*nodes SQL table*

Each row of the **graph** table represents an edge connecting two nodes from the **nodes** table. The columns of a graph edge are an ID for that edge; a node ID for the top (parent) of the edge; a node ID for the bottom (child) of the edge; and its type ("yes" or "no") indicating whether the child is the "yes" child or the "no" child of the parent. For example, the first row of the table shown below indicates that node 2 ("Is it hairy?") is the "yes" child underneath node 1 ("Does it have 4 legs?").

graphid	parentid	childid	type
101	1	2	yes
102	1	7	no
103	2	3	yes
104	2	6	no
105	3	4	yes
106	3	5	no
107	7	8	yes

108	7	9	no
-----	---	---	----

*graph SQL table*

### 3) Firebase remote database:

We provide a data set that you can connect to [Firebase](#) as a JSON file. You can access Firebase either by setting up Firebase yourself (recommended), or by using our shared Firebase database that is accessible to all students (less recommended; better as a backup option).

#### a) Setting up Your Own Firebase Database:

You can set up Firebase for your Google account and attach it to your project as described in the [Firebase lecture slides](#) and in [Google's Firebase setup tutorial](#). Download our [animalgame.json](#) file above and import that into Firebase, and then you can give yourself full read/write access to its data. Then you can "Import JSON" to upload our data into your personal Firebase database.

**Detailed steps to set up your own Firebase with our animal game data:** Go to <https://firebase.google.com/>. Click "Get Started." Then "Add Project." Follow the steps there to add your project. Once the project is added, you'll get to the project's "Overview" page. From there, on the left side click the "Database" list option. Now on the right it should say "Realtime Database". On the right side there are icons with a +, -, and "...". Click the latter "..." icon, and a menu will pop up. Choose "Import JSON." Browse to your [animalgame.json](#) file. When it's done importing, you should now have a Firebase database set up to use in your project. You will also need to download a [google-services.json](#) file and put it in the **app/** folder of your project.

**Common error:** If you are trying to create your project in Firebase and see a red error box saying, "There was an unknown error while processing the request. Try again.", it is likely because you are using the wrong account. Log out of your [@stanford.edu](#) email account and log in using your [@gmail.com](#) Google account and try again.

The data in the Firebase database is similar to the **SQL database** described previously, so please read that information before reading this information. The Firebase data is essentially the key/value map equivalent of the SQL nodes table described on the last page, with yes and no ID values added to each node for convenience. The following JSON snippet illustrates the general structure of the Firebase data:

```

1 "animalgame_small" : {
2   "nodes" : {
3     "1": {
4       "no" : 7,
5       "id" : 1,
6       "text" : "Does it have 4 legs?",
7       "type" : "question",
8       "yes" : 2
9     },
10    "2": {
11      "no" : 6,
12      "id" : 2,
13      "text" : "Is it hairy?",
14      "type" : "question",
15      "yes" : 3
16    },
17    ...
18  }
19 }
```

*subset of [animalgame\\_small](#) Firebase data*

So for example, to get the string "Does it have 4 legs?" out of the data set, you would visit the node with the path of ["animalgame\\_small/nodes/1/text"](#).

In the SQL data, the graph table represents the "yes" and "no" connections between parent and child nodes. But in the Firebase data, this information is stored directly within the nodes children themselves. For example, the path ["animalgame\\_small/nodes/1/yes"](#) stores the ID of the "yes" child of node 1, and the path ["animalgame\\_small/nodes/1/no"](#) stores the ID of the "no" child of node 1. A node whose type is "answer" will not have any "yes" or "no" child.

Note that you can replace [animalgame\\_small](#) with [animalgame](#) above to use the full large data set. Please write and debug your code using the small data set to reduce load on the server; test the large data set only after you have verified your small version thoroughly.

In the starter code we provide a small class named DBNode that correspond to the structures shown in the figures above. These may help you if you use Firebase's `getValue(DBNode::class.java)` method.



## b) Shared Firebase Database (read-only):

If you are unable to get your own Firebase database set up and working, we also provide a shared publicly readable Firebase database of our own at the following URL. Note that in order to use our shared Firebase successfully in your project, you must set it up as shown in our Firebase lecture slides, including modifying your **build.gradle** files, downloading a **google-services.json** file (see Starter Files above), and you must put your Kotlin class files in the package named `cs193a.stanford.edu.animalgame`. The following are the URL, user name, and password for the shared data set. If these don't work, you can also try signing in anonymously.

- database URL: <https://animalgame-2fcb3.firebaseio.com/>
- username: `cs193a@stanford.edu`
- password: `csroxx`

Since all students are able to access the shared Firebase data set, and to avoid the possibility of contaminating or deleting the data, the Firebase is read-only. You can write the code to play the game, but you cannot add new nodes of data to the computer's data set. Being able to write new nodes into the database is one benefit of setting up your own Firebase rather than using the shared one.

**Caution:** If you develop using Firebase, please keep in mind that the service allows only a limited number of connections and queries. If your app is poorly written and makes a huge number of queries, it could cause the service to exceed its limit, blocking the service for you or others. Please write your code carefully, performing a single query and printing its results, then aborting, to make sure your code is behaving the way you expect. Please do not write queries that attempt to download the entire database as a [DataSnapshot](#); instead, query only to get the single node you need at any given time.

## Extra (Optional) Features:

You are not required to add any special features to this program. But if you are interested in adding more functionality to your app beyond what is required, here are some options:

- **Talking game:** Use [TextToSpeech](#) features to make your game speak to the user during gameplay.
- **Loading animation:** Web APIs can be slow, and it's unpleasant to the user if there is no indication that the app is loading. Display an animation or "loading" screen while the data is in transit. If you want to simulate a slow network connection, add **&delay=10** to the end of any REST API URL to add a 10-second delay.
- **Additional versions of game:** Build/use another data set that is not about animals.
- **Better graphics:** Find new header / title screen images. Perhaps find images of the animals themselves so that the animal can be displayed when it is guessed by the user.
- **Use more than one dataset:** Make your code flexible so that it could play the game using more than one of the three data sources. You could provide a flag in the code to switch data sources, or an option in the settings activity, to choose between them. Doing this extra feature will give you a lot of practice dealing with the different types of data!

## Style Requirements:

Here are 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.

- **Query intelligently:** In your code that queries a data source, you should take care to filter the data properly in your query. For example, if you are doing an SQL query, instead of just saying, `SELECT * FROM graph`, you should filter out columns you don't need instead of saying `*`, and you should include a `WHERE` clause to greatly reduce the number of rows sent back from the database. Similarly, if you query Firebase, you should not just download the entire DB all at once and loop over it; instead, you should fetch individual node items and look at them, using the IDs in them to know which node to fetch next as needed.
- **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. You do not need to use any specific comment format, nor do you need to document exactly what each parameter or return value does. 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 5
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. Our lecture code from Week 1 shows examples of reducing fields by accessing values from widgets instead (using `getText` and similar). (We consider it okay to declare widgets as fields if you use the ButterKnife library and its `@BindView` annotation.)
  - **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.*