Android UI Workshop: Build an Interactive Quiz App

by Jessica Thornsby 8 Jan 2013

Difficulty: Beginner Length: Medium Languages: English

Android SDK | Mobile Development

This post is part of a series called Learn Android SDK Development From Scratch.

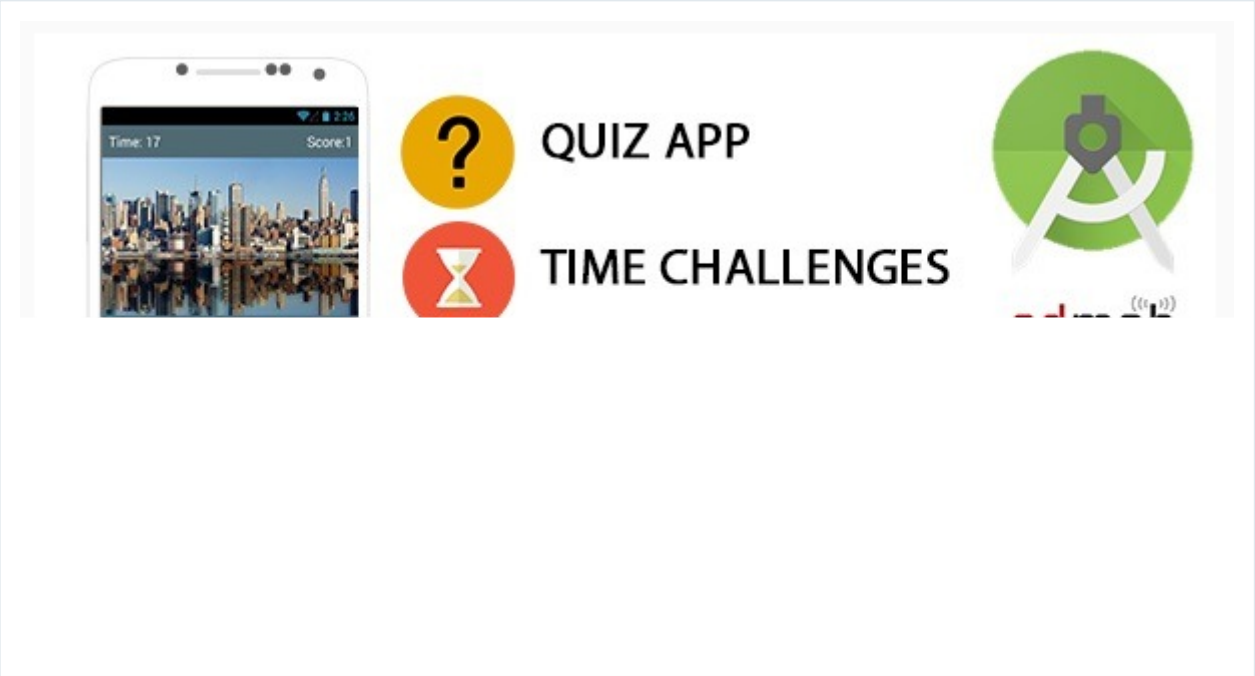
Build an ASCII Art Editor: Interface Setup

Get a hands-on crash course in building an effective and attractive UI for your Android app, and design an interactive quiz app along the way!

Looking for a Shortcut?

Looking for a Shortcut?

In this tutorial, we'll build a quiz app from the ground app, but if you're looking for a quicker solution, check out Envato Market's Quiz App Template for Android. It's a well-designed template that you can customize as you wish to create your own quiz app from a solid foundation.



If you'd prefer to build it from scratch by yourself, read on for the instructions.

Step 0: Getting Started

For the Android app developer, user interface (UI) is key. Your app may boast cutting-edge functionality, but if it’s wrapped in an unappealing UI it’s unlikely that users will stick around long enough to discover just how great your app really is.

This tutorial provides a hand's on, crash course in building an effective UI, using the example of a simple quiz app. The app poses a question and presents the user with four possible answers, implemented as ImageButtons. 'Correct' and 'Incorrect' messages are displayed via a popup, and the user can see at a glance which version of the app they're running.

As you build the app, we'll cover the following:

- RelativeLayout
- Designing for multiple screens
- Adding and leveraging image resources
- String resources
- App accessibility
- Creating stylish text
- Custom XML Styles
- Making buttons 'tappable'
- Toast notifications

Start by creating an Android project called 'AndroidGame' and a new Activity (we'll stick with the default 'MainActivity' filename.) Delete the 'Hello World' TextView that Eclipse generates automatically, along with any default parent view that has been added.

EDITOR'S NOTE: This tutorial is based on the Vectortuts+ artwork created in the tutorial "How to Create a Pub Chalkboard UI in Adobe Illustrator" by Sharon Milne.

Step 1: Define a RelativeLayout

A RelativeLayout is a flexible layout that allows you to organize UI elements in relation to one another, making it particularly useful for developers who need fine-grained control over the exact positioning of their UI components.

Create a basic RelativeLayout view group to hold all your UI elements:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >


  </RelativeLayout>
```

Step 2: Designing for Multiple Screens: Background Image

As an open source operating system, Android crops up on all sorts of devices, and is the OS of choice for many different manufacturers - it's no longer even restricted to smartphones! This presents both exciting opportunities and headache-inducing challenges for Android developers. When developing your UI, keep in mind that your app needs to display correctly across as many devices as possible. While designing for multiple screens is too broad a topic for this tutorial, as a starting point you should provide alternate versions of every image used in your app. These alternate images should be optimized for screens with different pixel densities. When your app launches, Android will automatically load the correct image based on the device's display characteristics.

Open the 'res' folder and you will see four drawable folders that Eclipse generates by default:

- drawable-ldpi

- drawable-mdpi
- drawable-hdpi
- drawable-xhdpi

These folders target screens with different pixel densities, following a 3:4:6:8 scaling ratio (for example, if drawable-ldpi contains an image of 36 x 36 pixels, then drawable-mdpi should contain an image of 48 x 48 pixels, and so on).

Create four versions of your background image based on this ratio, and drag each version into the appropriate folder.

Add the background image to RelativeLayout using the 'android:background' attribute, referring to the image resource as you would any drawable object:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/background" >

</RelativeLayout>
```

Boot up the Android Emulator and take a look at your work so far.

Step 3: Creating ImageButtons

The first step of creating an ImageButton is to add the required images to your project. Again, because you want your device to display correctly across a range of screens, you should provide four versions of each image. We'll start with the '2003' button, so drag and drop the four versions of your '2003' image into the corresponding folders.

ImageButtons are created using the 'ImageButton' tag followed by the usual height/width attributes, and some RelativeLayout-specific alignment attributes.

Create your first ImageButton and tweak the alignment until you're happy with how it's positioned. In this example, I'm using the following:

```
<ImageButton
        android:id="@+id/imageButton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true">
```

But your ImageButton is missing the most important part: an image! Refer to this image as a drawable resource, but this time using the 'android:src' property:

```
<ImageButton
        android:id="@+id/imageButton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:src="@drawable/button2003" />
```

Check how this appears in the emulator.

Repeat this process for the other four buttons. This is a good opportunity to experiment with the various RelativeLayout attributes (the Android docs are a great source of additional information on the different layouts.)

As you fine-tune your layout, you may find it useful to keep the emulator open, switching between it, the graphical layout editor and your XML.

Here's the code I'm using:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/background" >
```

```
06
07      <ImageButton
08          android:id="@+id/imageButton1"
09          android:layout_width="wrap_content"
10          android:layout_height="wrap_content"
11          android:layout_centerHorizontal="true"
12          android:layout_centerVertical="true"
13          android:src="@drawable/button2003" />
14
15      <ImageButton
16          android:id="@+id/imageButton2"
17          android:layout_width="wrap_content"
18          android:layout_height="wrap_content"
19          android:layout_below="@+id/imageButton1"
20          android:layout_centerHorizontal="true"
21          android:src="@drawable/button2004" />
22
23      <ImageButton
24          android:id="@+id/imageButton3"
25          android:layout_width="wrap_content"
26
27          android:layout_height="wrap_content"
28          android:layout_below="@+id/imageButton2"
29          android:layout_centerHorizontal="true"
30          android:src="@drawable/button2005" />
31
32      <ImageButton
33          android:id="@+id/imageButton4"
34          android:layout_width="wrap_content"
35          android:layout_height="wrap_content"
36          android:layout_below="@+id/imageButton3"
37          android:layout_centerHorizontal="true"
38          android:src="@drawable/button2006" />
39
    </RelativeLayout>
```

The buttons may be displaying correctly in the emulator, but Eclipse is throwing multiple "Missing contentDescription attribute on image" errors. What's this all about?

Step 4: Making Your App Accessible

The ContentDescrition attribute makes applications more accessible to users who cannot easily consume graphical content, by providing an alternate text description. This description can be used by screen readers and similar accessibility tools.

The first step is to define some new string resources. Strings are defined in the res/layout/strings.xml file, so open it and switch to the strings.xml tab if necessary.

String resources are created with the following syntax:

```
<string name="name">Text to display</string>
```

Define a string for the ContentDescription of each ImageButton:

```
<string name="button2003">Answer 2003</string>
<string name="button2004">Answer 2004</string>
<string name="button2005">Answer 2005</string>
<string name="button2006">Answer 2006</string>
```

Return to your activity_main.xml file and add a ContentDescription attribute to each ImageButton, e.g:

```
android:contentDescription="@string/button2003"/>
```

Step 5: Creating Stylish Text

In reality, it's unlikely you'll cease developing an app at Version 1.0, as you cook up new content and release fixes for any pesky bugs that users discover. Therefore, it's handy if people can tell at a glance what version of your app they're running. The easiest way to do this is with a TextView.

Open the strings.xml file and define the current version number as a string:

```
<string name="Version">Version 1.0</string>
```

You can quickly create a TextView that calls "@string/Version" but Android's default text style isn't particularly appealing. Thankfully, TextView has a range of attributes that can give boring old text some pizazz. You can read more about all the available attributes at the Android docs, but in this tutorial we'll use the following:

the Android docs, but in this tutorial we'll use the following:

- **android:textColor** - sets the colour of your text. It can either be defined as a string resource, or as a colour value (for more information on values, see the Android Docs: Colour section.)
- **android:textSize** - sets the text size, which can be given either as px (pixels) sp (scaled pixels) dp (density-independent pixels) in (inches) or mm (millimeters)
- **android:textStyle** - either italic or bold.

Create your TextView, add some style attributes, and then reference the “Version” string:

```
<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"

    android:layout_alignParentLeft="true"
    android:layout_marginLeft="26dp"
    android:paddingBottom="5dp"
    android:text="@string/Version"
    android:textColor="#f7f9f6"
    android:textSize="13dp"
    android:textStyle="italic" />
```

This works, but if you’re planning to re-use this set of attributes elsewhere then you should define it as a custom "Style." A style is a time-saving collection of properties (in this case font colour, font size, and the italic style) that specifies the appearance of a view, or even an entire application.

You’ll often hear ‘styles’ and ‘themes’ mentioned in a similar context. Essentially, the only difference between a style and a theme is their scope: ‘styles’ that are applied to activities become ‘themes.'
To define a custom style:

1. Open the res/values/styles.xml file, ensure it has opening and closing ‘resource’ tags, and give your style a unique name:

and give your style a unique name.

```
<resources>
     <style name="VersionFont">
     </style>
</resources>
```

2. Define all the attributes that belong to this style:

```
<item name="android:textSize">13dp</item>
```

For this tutorial, we'll define the following custom style:

```
<resources>
  <style name="VersionFont">
      <item name="android:textColor">#f7f9f6</item>
      <item name="android:textSize">13dp</item>
      <item name="android:textStyle">italic</item>
  </style>
</resources>
```

Applying this custom style to your TextView is easy, simply replace all the attributes defined by the VersionFont style, with: `style="@style/VersionFont"`.

e.g:

```
<TextView
   android:id="@+id/textView1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_alignParentBottom="true"
   android:layout_alignParentLeft="true"
   android:layout_marginLeft="26dp"
   android:paddingBottom="5dp"
   android:text="@string/Version"
   style="@style/VersionFont" />
```

This custom style isn't defined in the Android namespace, which is why we don't need to include an "android:" reference.
Both of these methods are valid ways of styling text, but keep in mind that a custom style can speed up the coding process if you're going to re-use the same collection of attributes across your application.

Step 6: Making Your App Interactive

Your UI may be finished, but there's one final step: letting the user know whether they've selected the correct answer. The easiest way to make buttons 'tappable' is by implementing the `android:onClick` property for each button.

We'll start by adding it to the first button:

```
<ImageButton
        android:id="@+id/imageButton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:src="@drawable/button2003"
        android:contentDescription="@string/button2003"
        android:onClick="onClick2003"   />
```

The name of the onClick method must correspond to a public method in the MainActivity.Java file, so add the following to your Activity:

```
public void onClick2003 (View view)
```

Now specify the behaviour that "onClick2003" triggers. A straightforward way of telling the user they've got the question wrong, is with a toast notification:

```
public void onClick2003 (View view)
{
    Toast.makeText(this, "Wrong! Try again", Toast.LENGTH_SHORT).show();
}
```

Run this updated code in the emulator and get clicking to check your toast is firing properly.

Success!

Success!

Add `onClick` to the other three buttons and create corresponding public methods in your Activity, with appropriate messages attached. Once you've finished, boot up the emulator and have a click around to check everything's working okay.

Conclusion

Over the course of this tutorial you've learned the essentials of Android UI design and created a simple, but effective UI that has some working functionality to boot! Try applying

some of these techniques to your own Android projects, to ensure your UI doesn't let your app down.

Jessica Thornsby

Jessica Thornsby

Jessica Thornsby is a technical writer based in Sheffield. She writes about Android,

Eclipse, Java, and all things open source. She is the co-author of iWork: The Missing Manual, and the author of Android UI Design.

JessicaThornsby

Download Attachment

Translations

Envato Tuts+ tutorials are translated into other languages by our community members—you can be involved too!

Translate this post

Powered by

Vinodh Kumar Gokulan • 7 years ago

very good example of Quiz APP

40 ^ | v • Reply • Share ›

Matt • 7 years ago

Finally we have a tutorial about UI for Android! Please could write more tutorials on this topic? Thanks and congratulations! Really appreciated.

15 ^ | v • Reply • Share ›

pradeep • 6 years ago

Can you please share the source code of the app ?

14 ^ | v • Reply • Share ›

Guest • 6 years ago

Nice friends... I ask for your permission to use this as a base for my beta apps to help child learn quran...

8 ^ | v • Reply • Share ›

sandeepbh • 7 years ago

This is good tutorial. Now pls. guide us on how to manage an application where 1000 questions are there in the quiz. Do we need to create 1000 activities.

10 ^ | v 1 • Reply • Share ›

cioalex ➚ sandeepbh • 7 years ago

Normally you would use a Database if you have so many questions. And instead of defining specific onClickListener you would manage the text and the clicked events inside your Activity dynamically. Might work easier. Even its interesting to see this way of implementing an Eventlistener.

6 ^ | v • Reply • Share ›

Nipun Mohta ➚ cioalex • 7 years ago

Hi! I also want to make an app with more than a 1000 questions and also want to categorize those questions so that the user can pick on what topics does he/she want to attempt the quiz. Also, I want to add a mode where user needs to the answer a question in say 1 min or 30 sec and the sooner he/she answers the better their score and so on for lets say a 15 min quiz session. And, in the end they can post their scores on various social media websites. Have researched about the posting part. But, I have no idea how to make such an online database and then make it interact with the android app. Also, would like to display tiny facts after the question and link to a website to read more about the topic in the question. Please guide me/point me towards such tutorials, how all this can be achieved.

10 ^ | v 1 • Reply • Share ›

Ray ➚ Nipun Mohta • 6 years ago • edited

i want to create an App such as this one...do u have an update about this?

1 · Reply · Share ›

zhenyuan ➚ Nipun Mohta • 4 years ago

Hi how is your progress? i would like to make an app as you described. Any help would be kindly appreciated. Thanks

· Reply · Share ›

langosta ➚ Nipun Mohta • 5 years ago

have you found how to do this. i would like an app like the one you described, if you find anything please let me know.

· Reply · Share ›

QuizVizag • 7 years ago

It is an Excellent and lucid tutorial that I have ever gone through, i am an enthusiast, kindly help in sorting out the following three errors faced at Main activity.java,thanks in advance.

```
1 package com.example.quizgamesudha;

2 import android.os.Bundle;
3 import android.app.Activity;
4 import android.view.Menu;

5 public class MainActivity extends Activity {

6    @Override
7    protected void onCreate(Bundle savedInstanceState) {
8        super.onCreate(savedInstanceState);
9        setContentView(R.layout.activity_main);
10    }


11    @Override
12    public void onClick2003(View view)
13   {                Toast.makeText(MainActivity.this, "Wrong! Try again", Toast.LENGTH_
14    }

15 }
```

Errors: 1) Error at line No.13 Toast cannot be resolved
2) Error at line No.13 Toast cannot be resolved to a variable
3) Error at line No.12 View cannot be resolved to a type

1 · Reply · Share ›

Warren Lebovics ➚ QuizVizag • 6 years ago

Hi QuizVizag,

Do you have toasts imported?

Your toast on line 13 looks okay, but I prefer to substitute "MainActivity.this" with "getApplicationContext()". This works in my apps.

As for the Line 12 issue, I usually like to set my buttons up like so:

Define the button above the onCreate() method:

//Start Code//

//Start Code//
Button btnName;
//End Code//

Then use the following code with the onCreate() method:

// Start Code//
btnName = (Button)findViewById(R.id.<yourbuttonidfromlayout>);
btnName.setOnClickListener(new View.OnClickListener() {

see more

1 • Reply • Share ›

sandeep ➚ QuizVizag • a year ago

you have get the widget or perticular button from main.xml to main activity like this Button btn = (Button) findViewById(R.id.btn2003);

• Reply • Share ›

Vishi ➚ QuizVizag • 3 years ago

you can do changes in onClick event sometime it would'nt be acceptable .. :)

• Reply • Share ›

Alexander ➚ QuizVizag • 6 years ago

This is amazing but I am unable to complete the project, would it be possible to send me a .zip file by email? Many thanks

my email is s_ander26@hotmail.co.uk

Sandy

• Reply • Share ›

Mystic • 6 years ago

nice friend... allow me to use this as a base for my apps to help kids learning Quran....

1 • 1 • Reply • Share ›

R SINGARAM • 6 months ago

How do we store user responses of this app on Microsoft Azure?

• Reply • Share ›

anukriti goel • 2 years ago

Quiz APP code you write complete in one step

• Reply • Share ›

anukriti goel • 2 years ago • edited

Explore a shopping mobile app and a social networkingmobile app to identify the various navigation, table, form, search, filter, and sort design patterns used in the apps.

Prerequisite:To perform this exercise, you need to have a smartphone with the Internet connection. In addition, you need to download the apps, if the apps are unavailable on the

internet connection. In addition, you need to download the apps, if the apps are unavailable on the device.

Help this question

Reply • Share ›

shagun arora • 2 years ago

Great tips. used them to edit certain elements of my game check this out http://bit.ly/2vBy0YA

Reply • Share ›

funny • 2 years ago

See more details for mcq app in android studio with source code and essay to explain watch full video

see more

Reply • Share ›

funny • 2 years ago

See more details for mcq app in android studio with source code and essay to explain watch full video

Reply • Share ›

William Gosal • 2 years ago

Attachment cannot be downloaded.

Reply • Share ›

Villaman Noel • 2 years ago

Not the best application UI

Reply • Share ›

Imran Ali • 3 years ago

Miss Jessica Thornsby, how to set numeric data in radio button. I am creating an IQ based game in which 1 text view for display the question and 3 radio buttons for answers.

Reply • Share ›

Surinder • 3 years ago

Wow... This is really awesome for beginner. Cant believe I created interactive android App in just an hour. Please keep writing about it. Thank you!

Reply • Share ›

acmyers112 • 3 years ago

Thank you @Jessica

Reply • Share ›

AhmadHijazi • 4 years ago

Good!!
its a very good example for making quiz App

Reply • Share ›

Geovanny Morillo • 4 years ago

At last i found this

Reply • Share ›

Linus • 5 years ago

How to use database as a backend in the app?

Reply • Share ›

deepak k • 5 years ago

Good tutorial. Really appreciated. thanks

Reply • Share ›

Jasmine • 6 years ago

The app seems to be built in a way that if you wanted to add another question, you'd have to change all the images, etc. Is there an easier way to add questions?

Reply • Share ›

Pallav Sethiya ➚ Jasmine • 6 years ago

Dear, just pass the different question text to the textview each time the user answers correctly

Reply • Share ›

Calvin • 7 years ago

It's awesome but how do you add another level?

Reply • Share ›

Jquiu • 7 years ago

Excelent Finally .... Just Love it :D

Reply • Share ›

Tanish • 7 years ago

Great tutorial...keep the good work...hope we'll see more such ones..

Reply • Share ›

Santhosh Vijay • 7 years ago

awesome guys... Excellent UI Tutorial..

Reply • Share ›

Rex Budz • 7 years ago

Hi guys, the background used in this tutorial is really great. Where can i have those ? i want to use it in a tablet PC.