Grow with Google Challenge Scholarship
Lesson 12 Notes

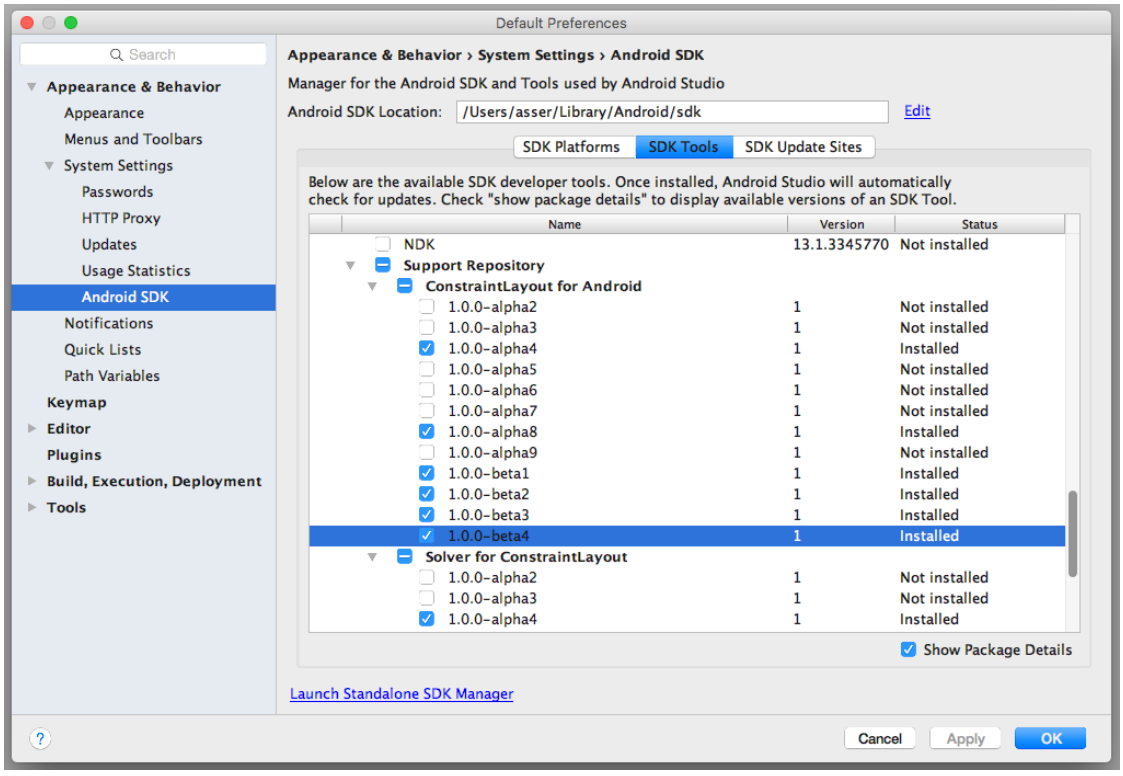
Installing the Constraint Layout Library

To continue building the Boarding Pass app, you'll need to make sure you have the latest Constraint Layout library. To ensure you have the latest Constraint Layout library:

1. Click **Tools** > **Android** > **SDK Manager**.
2. Click the **SDK Tools** tab.
3. Expand **Support Repository** and then check **ConstraintLayout for Android** and **Solver for ConstraintLayout**. Check **Show Package Details** and take note of the version you're downloading (you'll need this below).
4. Click OK.
5. Add the `ConstraintLayout` library as a dependency in your module-level `build.gradle` file:

```
dependencies {
        compile 'com.android.support.constraint:constraint-layout:1.0.0-beta4'
}
```

9. The library version you download may be higher, so be sure the value you specify here matches the version from step 3.
10. In the toolbar or sync notification, click **Sync Project with Gradle Files**.
11. Now you're ready to build your layout with `ConstraintLayout`.



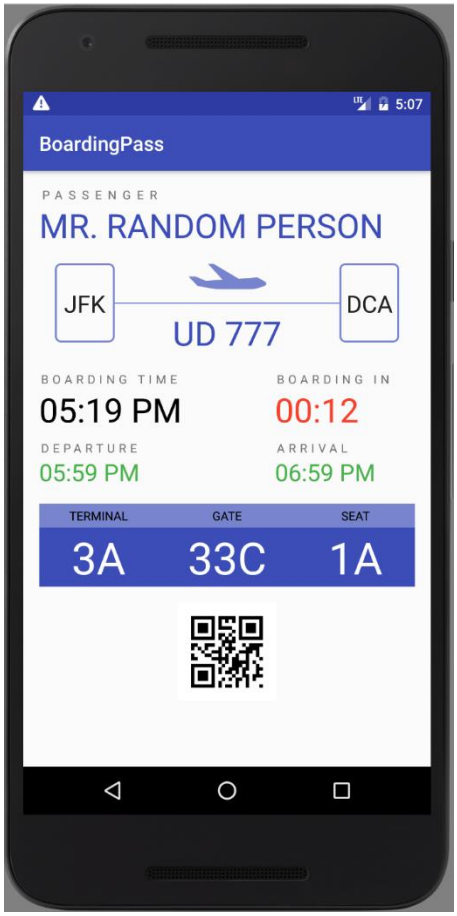
BoardingPass Code

In this lesson, we will work on building out a BoardPass app which helps users get information about their flight such as which terminal, gate number, and seat number.

The code for this app can be found in the **Lesson11-Completeing-The-UI** folder of the **Toy App Repository**.
If you need to a refresher on how the code is organized, please refer the **concept where we introduced the code flow**.

Explanation of BoardingPass App

Throughout this lesson, you will get to design this layout of a boarding pass app from scratch using Constraint Layouts and Data Binding, you will also get to add important accessibility and localization attributes in the app to cater for a wide diversity of potential users.



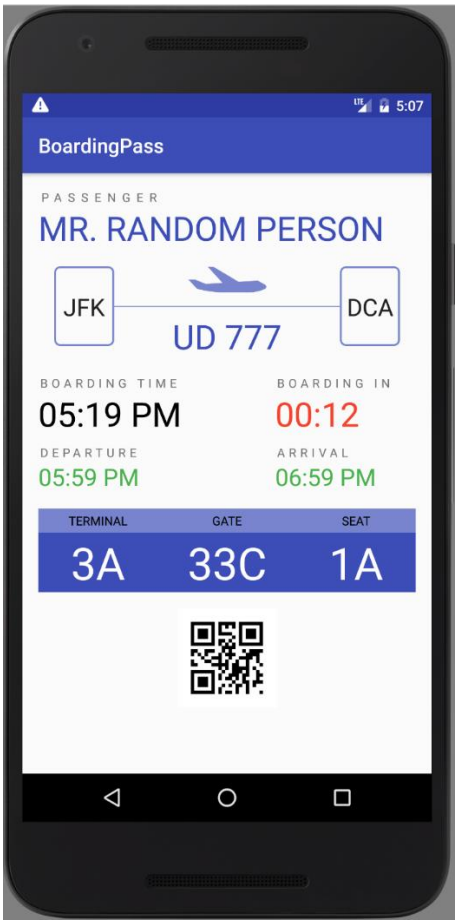
Accessibility refers to the design of products, devices, services, or environments for people who experience disabilities. Android provides accessibility features like

- **TalkBack** which is a pre-installed screen reader service provided by Google. It uses spoken feedback to describe the results of actions such as launching an app, and events such as notifications.
- **Explore by Touch** which is a system feature that works with TalkBack, allowing you to touch your device's screen and hear what's under your finger via spoken feedback. This feature is helpful to users with low vision.
- **Accessibility settings** that let you modify your device's display and sound options, such as increasing the text size, changing the speed at which text is spoken, and more. and more.

To take full advantage of those features, you should follow this checklist listed **here** from the android developer website, but let's focus on the first one for now which is the most important.

Describe user interface controls

If you look at our layout design for the boarding pass, you can immediately tell that our origin airport is JFK and the destination airport is DCA, simply because of their relative location on the screen and the direction of the plane. But if you cannot see this layout due to a visual limitation, simply having android read out everything on the screen won't tell you enough information about which is which.



That's why Android offers the `contentDescription` attribute to describe what any view actually presents, this description text is not displayed anywhere on the screen, but if the user enables accessibility services that provide audible prompts, then when the user navigates to that control, the text is spoken.

```
<ImageView android:contentDescription="@string/origin_label"/>
```

Ideally, in any app, you would want to describe all ImageViews, ImageButtons and all Checkboxes using the `contentDescription` attribute.

ContentDescription is just one of many things you need to consider when building your app for accessibility, others include:

- **Enable focus-based navigation** which makes sure users can navigate your screen layouts using external hardware like bluetooth keyboards.
- **No audio-only feedback** which guarantees any audio feedback to always have a secondary feedback mechanism to support users who are deaf or hard of hearing

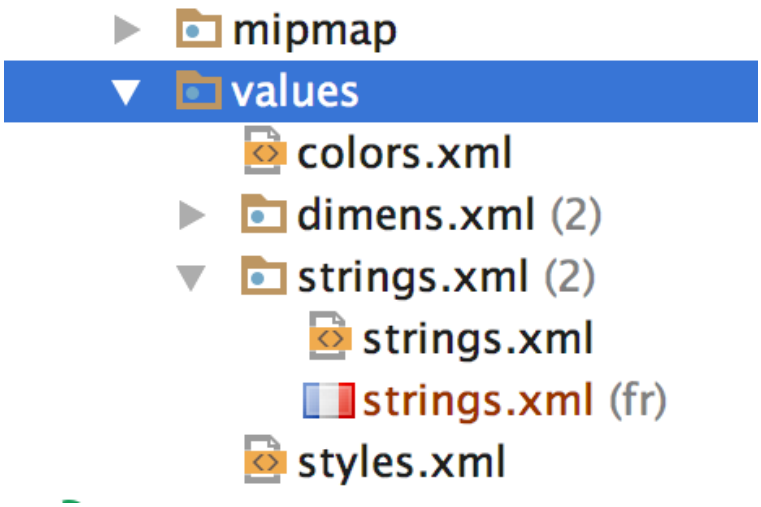
For the full check list and best practices follow this link: **Accessibility Developer Checklist**

Localization

Localization (also known as **Internationalization**) is the adaptation of a product or service to meet the needs of a particular language, culture or desired population's "look-and-feel".

I. Translation:

You should always design your app in a way that can be easily translated to other languages. To do so, any text that you would expect to be translated like labels and titles and button descriptions should all be defined as a string resource in `res/values/strings.xml`
This allows you to create other versions of `strings.xml` for other languages. This is done by creating a new `values` folder with the pattern `value-xx` where `xx` can be the abbreviation of any language from the ISO 639 code list **here**, for example `res/values-fr/strings.xml` will contain the french version of the strings.xml file with all the strings translated from the default language to french.
This way, when a user who has set up their phone to use french as the default language, android will automatically load the french version of strings and use all the pre-translated french labels.



Once a new values-xx folder is created, android displays all resource files grouped together like so.

Sometimes however, you would still want to use the strings recourses for strings that you don't intend to translate, this includes strings representing identifiers for views or variable names os string formats etc.

For those strings, there's an attribute called `translatable` that can be set to false to indicate that this string recourse should not be translated.

```
<string name="timeFormat" translatable="false">hh:mm a</string>
```

II. RTL support

If you're distributing to countries where right-to-left (RTL) scripts are used (like Arabic or Hebrew), you should consider implementing support for RTL layouts and text display and editing, to the extent possible.

You've already seen how to set image recourses to flip when RTL support activated to indicate the correct direction of travel using

```
<vector android:autoMirrored="true">  </vector>
```

Another set of attributes related to RTL support are the

```
android:layout_marginStart
android:layout_marginEnd
```

that correspond to

```
android:layout_marginLeft
android:layout_marginRight
```

respectively, but only when the default language is English (or any LTR language), for RTL languages however, **Start** is mapped to **Right** and **End** is mapped to **Left** instead, the idea is that when the app runs on a device with a default RTL language , everything will get mirrored by switching margins and constraints to the other side.

English LTR vs Arabic RTL screens of the same layout

Keep in mind that these Start and End attributes are relatively new, so to support older devices (prior to 4.1) you should still backup the Start and End margins with the outdated Left and

Right ones with the same values, and if your app ends up running on a more recent device the Left Right margins are ignored and the Start End ones are used instead.

This **Localization Checklist** offers some more important steps you should follow to make your Android app run on many devices in many regions and hence reach the most users.