Mobile Application Development
Week 13: Device Environments

With Android running on so many different size devices your app layouts must be flexible and adapt to support different size screens and orientations.

Screen compatibility
https://developer.android.com/guide/practices/screens_support
Android categorizes device screens using two general properties: size and density (the physical density of the pixels on the screen, known as DPI). Android generalizes the different configurations into groups that make them easier to target:

- Generalized sizes: small, normal, large, and xlarge.
- Generalized densities: mdpi (medium), hdpi (high), xhdpi (extra high), xxhdpi (extra-extra high), and others.

The screen's orientation (landscape or portrait) is considered a variation of screen size. Android automatically scales your layout in order to properly fit the screen. So your layouts for different screen sizes don't need to worry about the absolute size of UI elements but instead focus on the layout structure that affects the user experience (such as the size or position of important views relative to sibling views).

Flexible Layout
https://developer.android.com/guide/topics/large-screens/support-different-screen-sizes#flexible-layout
Constraint layout creates a layout responsive for different screen sizes.
Avoid hard coded layout values

- "wrap_content" tells the view to set its size to whatever is necessary to fit the content within that view.
- "match_constraint" tells the view to set its size to match the constraint.

Alternative layouts
If you can't define one layout that works for all screen sizes or orientations you'll need to define different layouts for each configuration orientation.
Remember that the default layout is in the res/layout folder.

Screen size qualifiers:
You can provide screen-specific layouts by creating additional res/layout/ directories and then append a screen configuration qualifier to the layout directory name for each screen configuration that requires a different layout

- layout-w600dp for screens that have 600dp of available width

You can also use the smallest width qualifier(sw) to provide alternative layouts for screens that have a minimum width measured in density-independent pixels

- layout-sw600dp for screens that have a minimum available width of 600dp

Orientation qualifier:
To provide different layouts based on the device's orientation you can add the "port" or "land" qualifiers to your resource directory names. Just be sure these come *after* the other size qualifiers.

- layout-land for devices in landscape orientation
- layout-sw600dp-land for devices with 600dp wide and larger and in landscape orientation

When an activity starts and a layout is inflated Android looks at the configuration and loads the appropriate resources. For a layout it will look to see if there's a layout that meets a specific configuration (orientation or screen size) and if so it will load that. If not it will use the layout in the res/layout directory. That's why on rotation the activity is killed and restarted, it has to handle the change in configuration and potentially inflate a different layout.

You can use other resource types and qualifiers for more specific configurations.
https://developer.android.com/guide/topics/resources/providing-resources

Supporting different languages
https://developer.android.com/guide/topics/resources/localization
We've been using IDs in our layout xml files and assigning those strings in our res/values/strings.xml file, which is the default strings file. If this default file is absent, or if it is missing a string that your application needs, then your application will not run and will show an error.

To add support for more languages, create additional values directories inside res/ that include a hyphen and the ISO language code at the end of the directory name. Android loads the appropriate resources according to the locale settings of the device at run time. You can do this for drawables too.

- values-es/ is the directory containing simple resources for the locale with the language code "es".

When an app starts Android selects which resources to load based on the device's locale.
If it finds a resource for that locale it will use the resources in it. If it doesn't it will look in the default res/strings.xml file. If it can't the resource in any file the app will crash.

Taco Adapt
The main layout in our taco app looks ok in landscape orientation because I added the scrollview, but let's look at how you define a new layout for landscape mode anyway.

Landscape variation
https://developer.android.com/studio/write/layout-editor.html#create-variant
Open the layout file in design or split mode.
Click the Orientation in Editor in the toolbar.
Create landscape variation
In res/layout/activity_main it creates a new layout file and opens it in the editor.
In Android view you see (land) next to it.
In the Project view you can see how this is really stored in app/src/main/res/layout-land
The title of the tab has the prefix "land/"
Edit this so you have a layout that looks good in landscape.
Be careful that you keep track of which file you're editing.
The changes you make here are only for this file. So it makes sense to do it at the end once the app is working because if you want to add new widgets or change anything for the existing widgets, you'll need to do it twice, in each file. They are in no way connected.

Here's what I changed:
I deleted the radio button group end constraint so it was no longer centered
I moved the check boxes to the right of the radio buttons by changing the top constraint to go to the textview (32) instead of the bottom of the radio group. I also changed the start constraint to go to the end of the radio group (32) instead of parent.

Then I moved the switch to the right of the spinner. When you're going to redo the constraints I find it easier to delete them, move the view, and create new constraints. I added a start constraint to the spinner and a top constraint to the radio group.
Then I moved the message text view to the right of the switch. Created a start constraint to the switch and an end constraint to the parent. Added a top constraint to a checkbox.
I moved both buttons to be in a row beneath the spinner, switch, and text view. I selected them both and created a horizontal chain. I also aligned their tops. For Create taco on the left I added a start constraint to the parent and for Find tacos I created an end constraint to the parent.
I didn't change anything for the review textview.

Run it and try it in both orientations.
Remember your two layout files are not connected in any way. So once you have multiple layout files if you make ANY changes or additions, you must do it twice.

Localization
In the Project view go to app/src/main/res right click New | Android Resource Directory values-es
(French fr, Spanish es, Japanese ja, German de. Other ISO language codes
http://www.loc.gov/standards/iso639-2/php/code_list.php)
Right click on your new folder New | Values resource file
strings.xml
Open your original strings.xml file and copy all the strings you have.
Go into your new es/strings.xml file and paste them in the <resources> tag.
Change the string values for your new language. (There are services that do this for a fee)

To test this in the emulator you need to change the language of the device.
Settings | System | Language & Input | Languages | Add a language
Add Espanol
Change the language to Espanol by reordering the languages.
This is done differently on different phones. Sometimes instead of System it's under personal.
(Android also lets you define a custom locale, but we shouldn't need that)
Now you'll notice that the apps that come with the phone, Settings, Camera, Phone etc are now all in the language you picked.

When you run your app you should see all your strings in the new language, but the text generated from Kotlin is still in English

To fix this we should make strings in our Kotlin file IDs and define the string in the xml file just as we did with our layout. In your strings files add

```
<string name="snackbarFillingMessage">Please select a filling
</string>
<string name="withFilling">with</string>
<string name="atLocation">at </string>
<string name="message">You\'d like</string>
<string name="tacoShopMessage">You should get tacos at</string>
```

and the Spanish version in the Espanol strings file.

```
<string name="snackbarFillingMessage">Seleccione un relleno </string>
<string name="withFilling">con</string>
<string name="atLocation">a </string>
```

```
<string name="message">Te gustaría</string>
<string name="tacoShopMessage">Deberías conseguir tacos en</string>
```

In our Kotlin files we can access a string resource using getString(R.string.*stringResourceName*)
MainActivity.kt

```
val fillingSnackbar = Snackbar.make(layoutRoot,
getString(R.string.snackbarFillingMessage), Snackbar.LENGTH_SHORT)

toppinglist = getString(R.string.withFilling) + toppinglist

val location = getString(R.string.atLocation) + " " +
spinner.selectedItem

messageTextView.text = getString(R.string.message) + "$filling tacos
$toppinglist $location"
```

TacoActivity.kt:

```
tacoShopName?.let {tacoShopTextView.text =
getString(R.string.tacoShopMessage) + " $tacoShopName"}
```

Now run your app in Spanish and all the text should be in Spanish.