Android

INTERACTIVE APPS - CHAPTER 2 OF THE TEXTBOOK

Introduction

We are going to create a Beer Adviser app in this lecture.

In our app, users can select the types of beer they enjoy, click a button, and get back a list of tasty beers to try out.

Here’s what you need to do

1. **Create a project.**

Create a basic layout and activity.

2. **Update the layout.**

Once you have a basic app set up, you need to amend the layout so that it includes all the GUI components your app needs.

3. **Wire the layout to the activity.**

The layout only creates the visuals. To add smarts to your app, you need to wire the layout to the Java code in your activity.

4. **Write the application logic.**

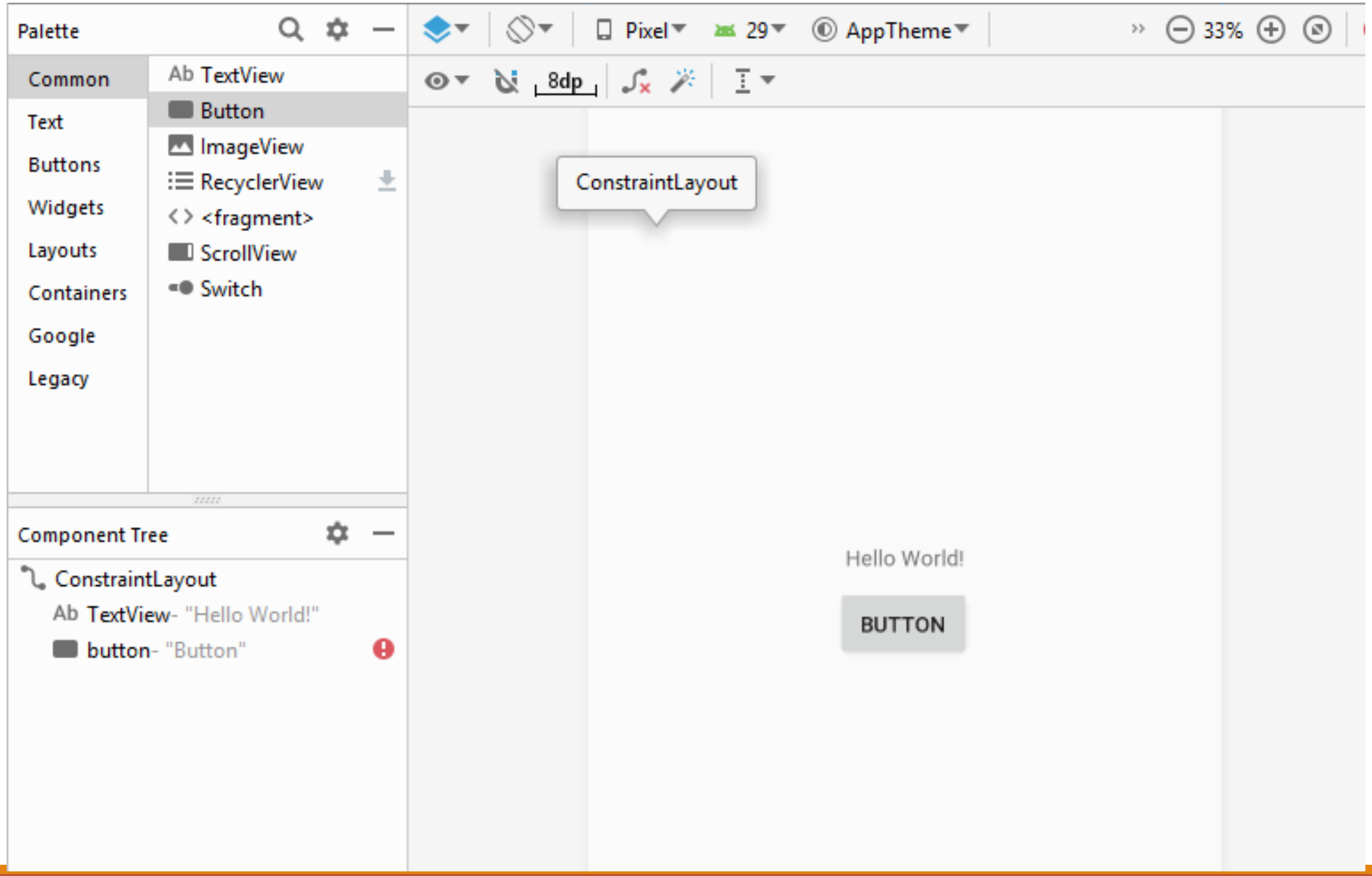
You’ll add a Java custom class to the app, and use it to make sure users get the right beer based on their selection.

Adding components with the design editor

There are two ways of adding GUI components to the layout: via XML or using the design editor. Let’s start by adding a button via the design editor.

To the left of the design editor, there’s a palette that contains GUI components you can drag to your layout.

If you look in the Widgets area, you’ll see that there’s a Button component. Click on it, and drag it into the design editor.


Palette
Common
Text
Buttons
Widgets
Layouts
Containers
Google
Legacy
TextView
Button
ImageView
RecyclerView
<fragment>
ScrollView
Switch
Component Tree
ConstraintLayout
TextView- "Hello World!"
button- "Button"
Pixel
29
AppTheme
33%
8dp
ConstraintLayout
Hello World!
BUTTON

Changes in the design editor are reflected in the XML

If you switch to the code editor, you'll see that adding the button via the design editor has added some lines of code to the file

```
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button"
        tools:layout_editor_absoluteX="161dp"
        tools:layout_editor_absoluteY="385dp" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Properties - I

Buttons and text views are subclasses of the same Android View class

here are some of the more common properties.

android:id

This gives the component an identifying name. The ID property enables you to control what components do via activity code, and also allows you to control where components are placed in the layout:

```
android:id="@+id/button"
```

android:text

This tells Android what text the component should display. In the case of <Button>, it’s the text that appears on the button:

```
android:text="New Button"
```

Properties - II

android:layout_width, android:layout_height

These properties specify the basic width and height of the component. "wrap_content" means it should be just big enough for the content:

```
android:layout_width="wrap_content"

android:layout_height="wrap_content"
```

Adding a Spinner

- Above the text View add the following code.
- In the design view position the Spinner above the text view

```
<spinner
    android:id="@+id/color"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

Spinner

Above the button we have a **spinner**. A spinner is the Android term for a drop-down list of values.

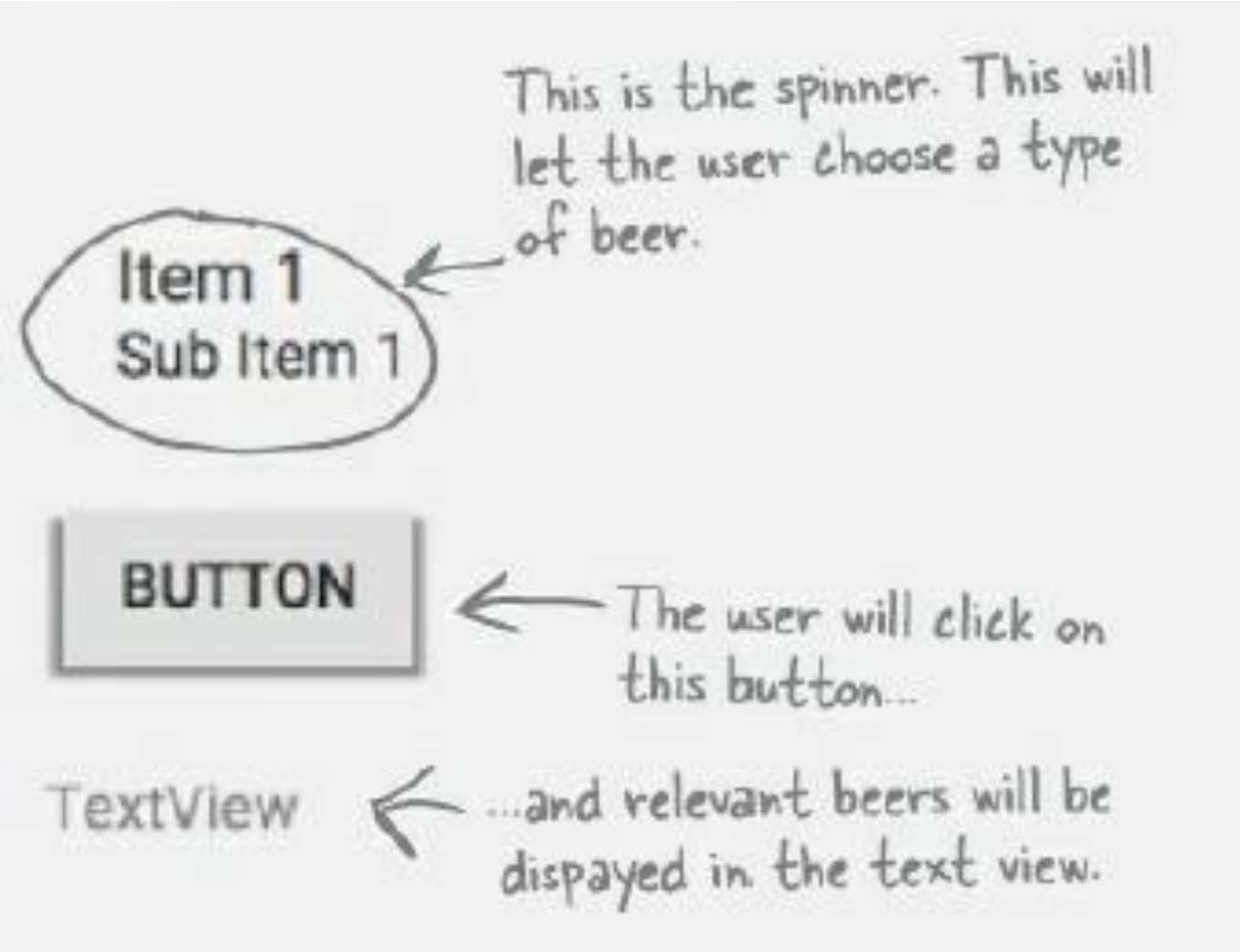
When you touch it, it expands to show you the list so that you can pick a single value.

A spinner provides a drop-down list of values. It allows you to choose a single value from a set of values.

GUI components such as buttons, spinners, and text views have very similar attributes, as they are all types of View. Behind the scenes, they all inherit from the same Android View class.

Final View

In the design view position the Spinner above the text view



Use string resources rather than hardcoding the text

At the moment, the button and text view both use hardcoded string values for their text properties.

It's a good idea to change these to use the strings resource file ***strings.xml*** instead. While this isn't strictly necessary, it's a good habit to get into.

Using the strings resource file for static text makes it easier to create international versions of your app, and if you need to tweak the wording in your app, you'll be able to do it one central place.

Add to string resources

Open up the *app/src/main/res/values/strings.xml* file. When you switch to the XML view, add the following:

```
<string name="app_name">Beer Adviser</string>
<string name="hello_world">Hello world!</string>
<string name="action_settings">Settings</string>
<string name="find_beer">Find Beer!</string>
<string name="brands"></string>
```

You need to remove the hello_world string resource, and add in two new ones called find_beer and brands.

Change the layout to use the string resources

Change the button and text view elements in the layout XML to use the two string resources we've just added.

Open up the *activity_find_beer.xml* file, and make the following changes:

Change the line **android:text="Button"**

To

android:text="@string/find_beer"

Change the line **android:text="TextView"**

To

android:text="@string/brands"

What we've done so far

1. **We've created a layout that specifies what the app will look like.**

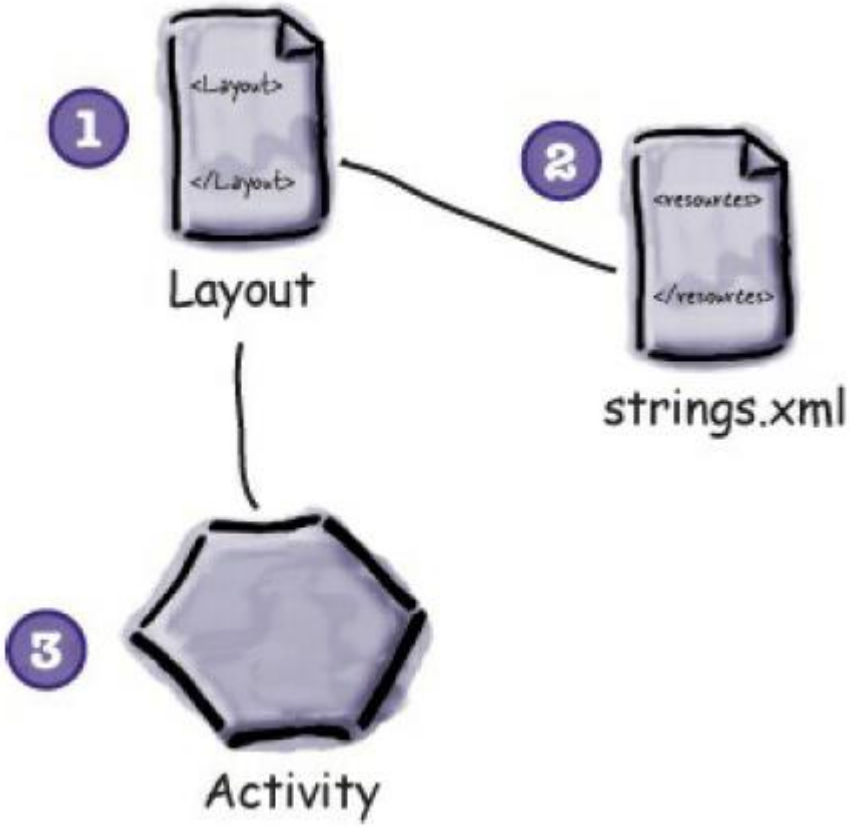
It includes a spinner, a button, and a text view.

2. **The file strings.xml includes the string resources we need.**

We've added a label for the button, and an empty string for the brands.

3. **The activity specifies how the app should interact with the user.**

Android Studio has created a basic activity for us, but we haven't done anything with it yet.



Add values to the spinner

We can give the spinner a list of values in pretty much the same way that we set the text on the button and the text view: by using a **resource**.

We need to specify an ***array*** of String values, and get the spinner to reference it.

Array resource

To add an array of Strings, you use the following syntax:

```
<string-array name="string_array_name">
    <item>string_value1</item>
    <item>string_value2</item>
    <item>string_value3</item>
    ...
</string-array>
```

This is the name of the array

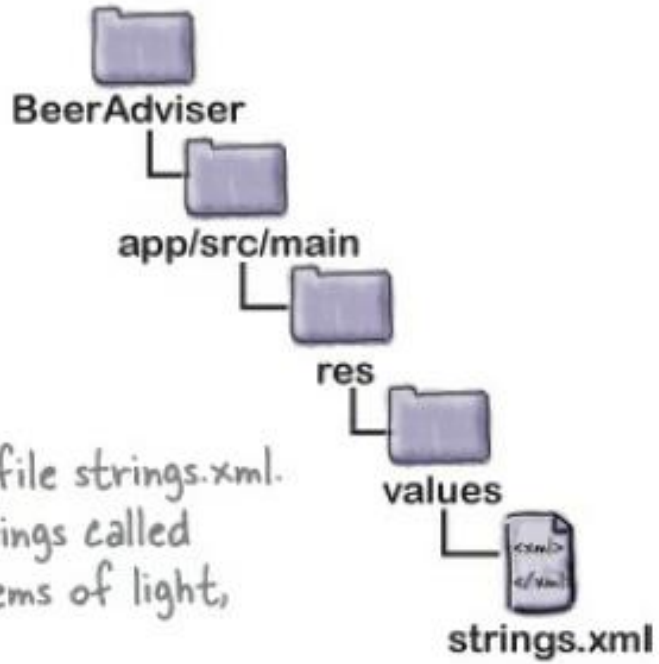
These are the values in the array. You can add as many as you need.

Adding the array to resources

Let's add a string-array resource to our app. Open up *strings.xml*, and add the array like this:

```
...
    <string name="brands"></string>
    <string-array name="beer_colors">
        <item>light</item>
        <item>amber</item>
        <item>brown</item>
        <item>dark</item>
    </string-array>
</resources>
```

Add this string-array to file strings.xml. It defines an array of strings called beer_colors with array items of light, amber, brown, and dark.



Reference a string-array

A layout can reference a string-array using similar syntax to how it would retrieve the value of a string. Rather than use

```
"@string/string_name"
```

Use @string to reference a string, and @array to reference an array.

```
"@array/array_name"
```

Get the spinner to reference a string-array

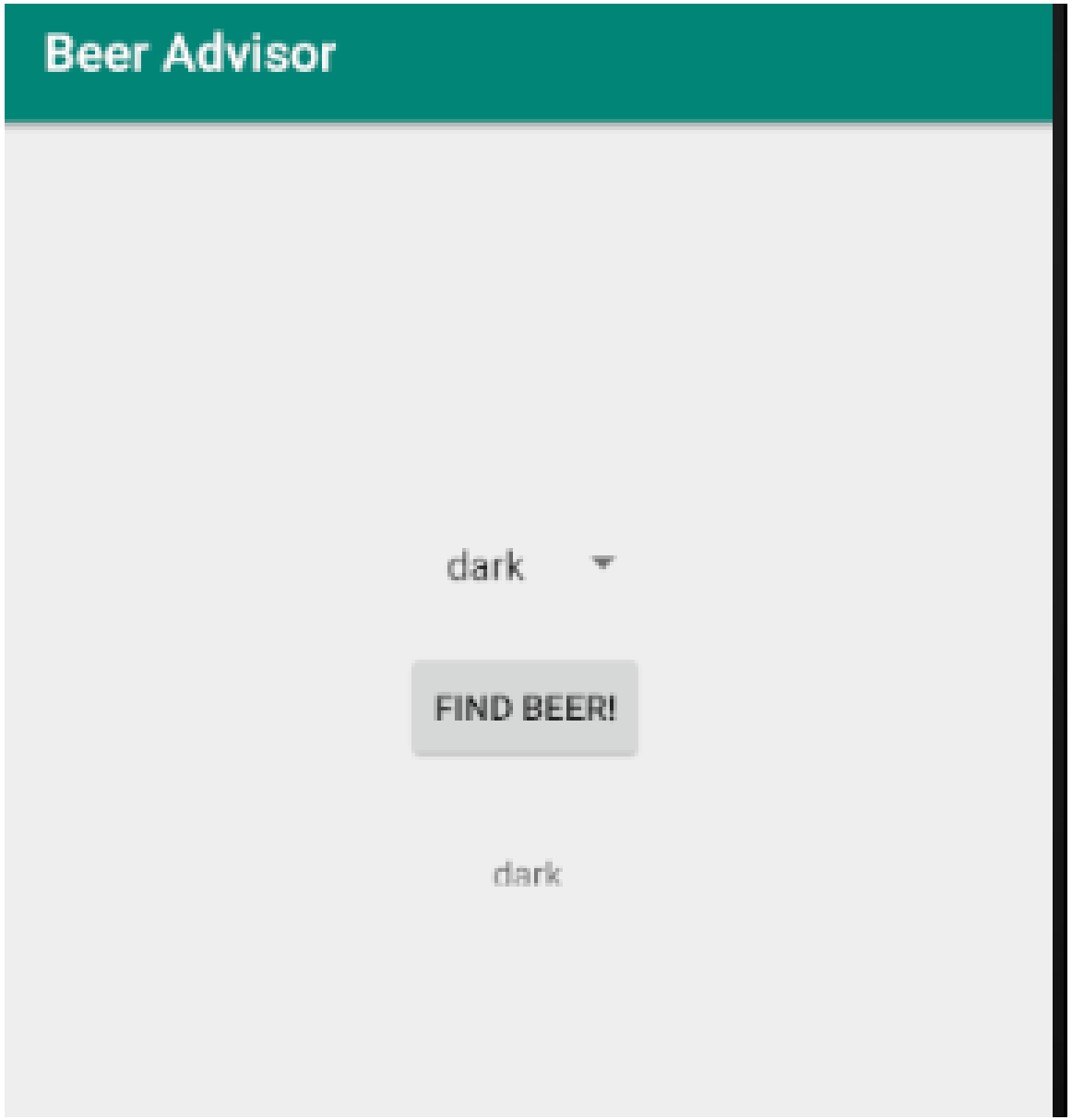
Let's use this in the layout. Go to the layout file *activity_find_beer.xml* and add an entries attribute to the spinner like this:

```
...
        android:entries="@array/beer_colors" />
```

This means "the entries for the spinner come from array beer_colors"

Run the app

You should get something like this



Make the Button Do Something

CODE

App Requirements

We want our app to behave something like this:

1. **The user chooses a type of beer from the spinner.**
2. **The layout specifies which method to call in the activity when the button is clicked.**
3. **The method in the activity retrieves the value of the selected beer in the spinner and passes it to the getBrands() method in a Java custom class called BeerExpert.**
4. **BeerExpert's getBrands() method finds matching brands for the type of beer and returns them to the activity as an ArrayList of Strings.**
5. **The activity gets a reference to the layout text view and sets its text value to the list of matching beers.**

Make the button call a method

Use onClick to say which method the button calls

Add an android:onClick attribute to the <button> element, and give it the name of the method you want to call:

```
android:onClick="method_name"
```

This means "when the compenent is clicked, call the method in the activity called method_name".

Add onClick

Go to the layout file *activity_find_beer.xml*, and add a new line of XML to the <button> element to say that method onClickFindBeer() should be called when the button is clicked:

```
android:text="@string/find_beer"
android:onClick="onClickFindBeer" />
```

When the button is clicked, call method onClickFindBeer() in the activity. We'll create the method in the activity over the next few pages.

Activity

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_find_beer);
    }
}
```

This is the onCreate() method. It's called when the activity is first created.

setContentView tells Android which layout the activity uses. In this case, it's activity_find_beer.

onClickFindBeer() method

On the previous page, we added an onClick attribute to the button in our layout and gave it a value of onClickFindBeer.

We need to add this method to our activity so it will be called when the button gets clicked.

This will enable the activity to respond when the user touches a button in the user interface.

Add an onClickFindBeer() to the activity

The onClickFindBeer() method needs to have a particular signature, otherwise it won't get called when the button specified in the layout gets clicked. The method needs to take the following form:

```
public void onClickFindBeer(View view) {
}
```

The method must be public.

The method must have a void return value.

The method must have a single parameter of type View.

If the method doesn't take this form, the method won't respond when the user touches the button. This is because behind the scenes, Android looks for a public method with a void return value, with a method

Add the onClickFindBeer() method below to your activity code:

We're using this class, so we need to import it.

```
...
import android.view.View;
```

The View parameter in the method refers to the GUI component that triggers the method (in this case, the button).

Add the onClickFindBeer() method to FindBeerActivity.java.

```
...
    //Call when the user clicks the button
    public void onClickFindBeer(View view){
    }
}
```

Use findViewById() to get a reference to a view

We can get a handle for our two GUI components using a method called **findViewById().**

The **findViewById()** method takes the ID of the GUI component as a parameter, and returns a View object.

You then cast the return value to the correct type of GUI component (for example, a TextView or a Button).

findViewById()

Here's how you'd use `findViewById()` to get a reference to the text view with an ID of brands:

We want the view with an ID of brands.

```
TextView brands = (TextView) findViewById(R.id.brands);
```

brands is a TextView, so we have to cast it as one.

What’s R?

R.java is a special Java file that gets generated by the Android tools whenever you create or build your app.

Android uses R to keep track of the resources used within the app, and among other things it enables you to get references to GUI components from within your activity code.

R is a special Java class that enables you to retrieve references to resources in your app.

Setting the text in a TextView

The TextView class includes a method called **setText()** that you can use to change the text property.

You use it like this:

```
brands.setText("Gottle of geer");
```

Set the text on the brands TextView to "Gottle of geer"

Retrieving the selected value in a spinner

You can get a reference to a spinner in a similar way to how you get a reference to a text view.

You use the findViewById() method as before, only this time you cast the result as a Spinner:

```
Spinner color = (Spinner) findViewById(R.id.color);
```

Retrieve the Currently Selected Item in the Spinner

This gives you a Spinner object whose methods you can now access. As an example, here's how you retrieve the currently selected item in the spinner, and convert it to a String:

```
String.valueOf(color.getSelectedItem())
```

This gets the selected item in a spinner and converts it to a String.

getSelectedItem()

The code

```
color.getSelectedItem()
```

actually returns a generic Java object. This is because spinner values can be something other than Strings, such as images.

In our case, we know the values are Strings, so we can use String.valueOf() to convert the selected item from an Object to a String.

We imported these classes

```
import android.view.View;
import android.widget.Spinner;
import android.widget.TextView;
```

We're using these extra classes.

Change the code to this:

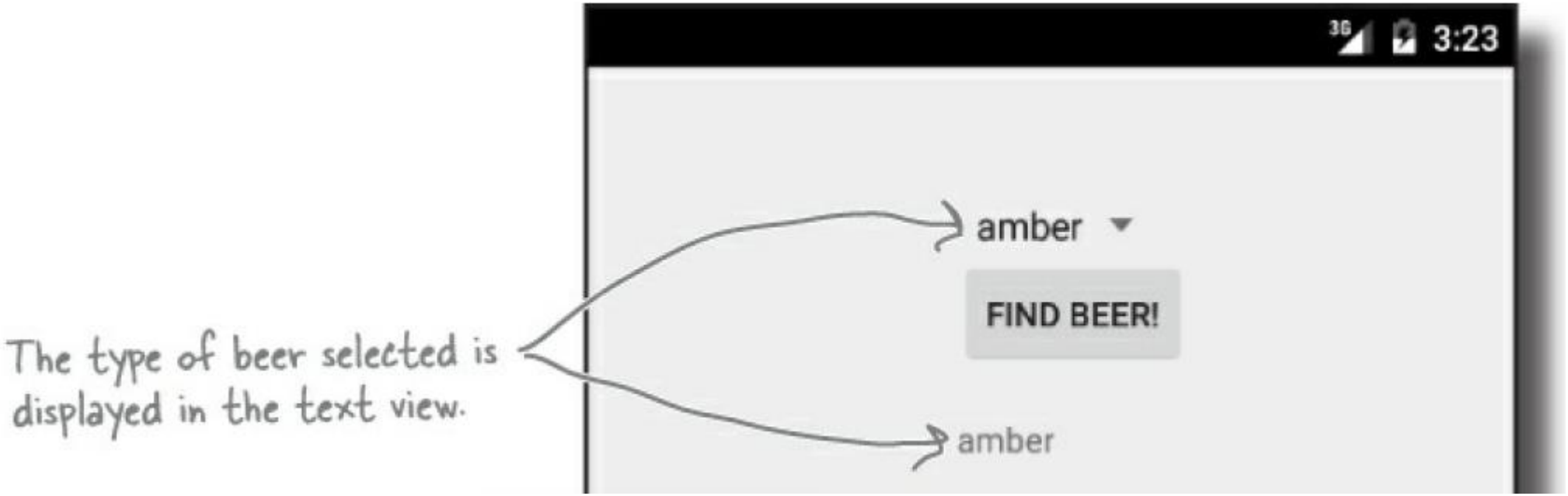
```
//Call when the button gets clicked
public void onClickFindBeer(View view) {
    //Get a reference to the TextView
    TextView brands = (TextView) findViewById(R.id.brands);
    //Get a reference to the Spinner
    Spinner color = (Spinner) findViewById(R.id.color);
    //Get the selected item in the Spinner
    String beerType = String.valueOf(color.getSelectedItem());
    //Display the selected item
    brands.setText(beerType);
}
```

findViewById returns a View. You need to cast it to the right type of View.

getSelectedItem returns an Object. You need to turn it into a String.

Test drive the changes

Make the changes to the activity file, save it, and then run your app. This time when we click on the Find Beer button, it displays the value of the selected item in the spinner.



Custom Java class

The Beer Adviser app decides which beers to recommend with the help of a custom Java class. The custom Java class is written in **plain old Java**, with no knowledge of the fact it's being used by an Android app.

Custom Java class spec

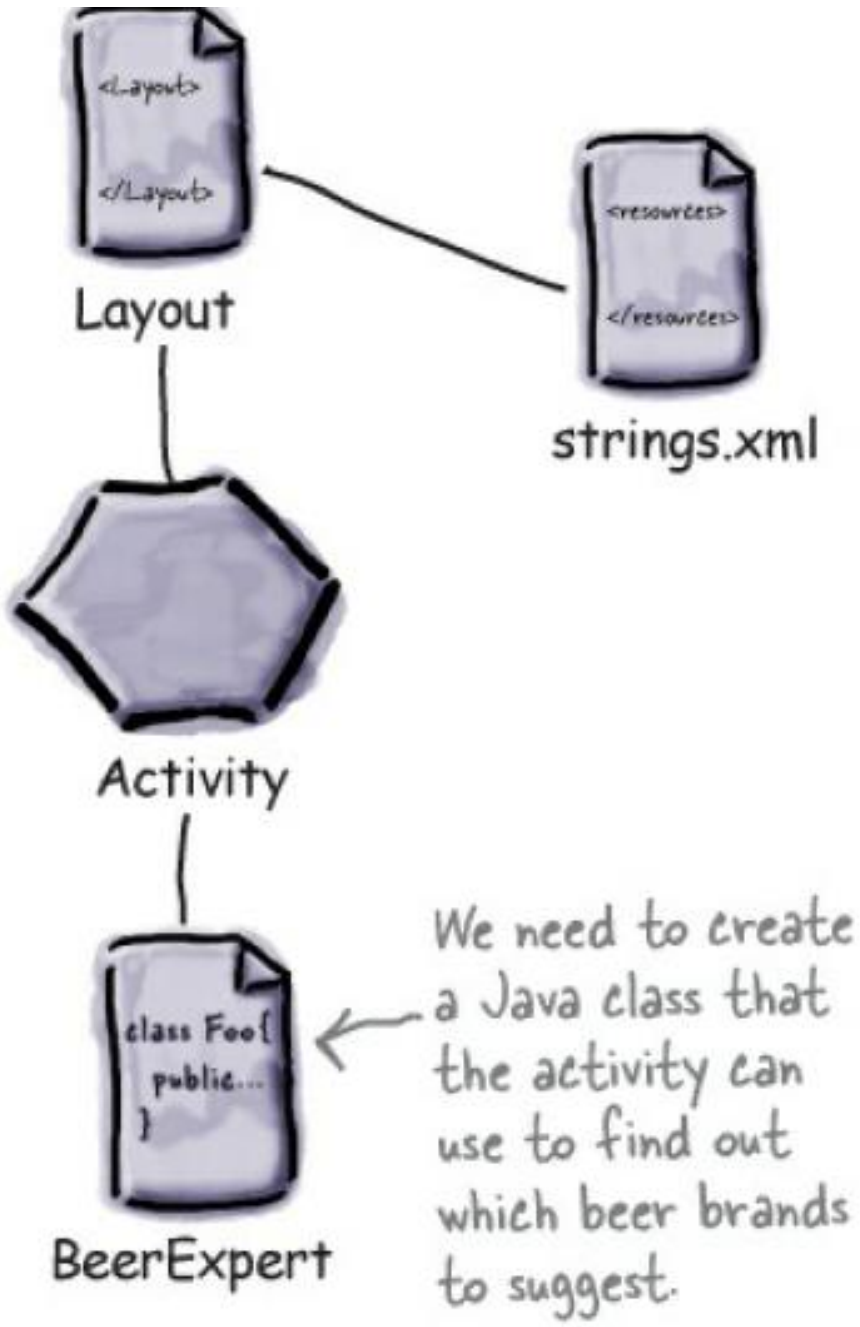
The custom Java class should meet the following requirements:

The package name should be com.hfad.beeradviser.

The class should be called BeerExpert.

It should expose one method, getBrands(), that takes a preferred beer color (as a String), and return a List<String> of recommended beers.

BeerExpert class



Building the custom Java class: BeerExpert

```
List<String> getBrands(String color) {
    List<String> brands = new ArrayList();
    if (color.equals("amber")) {
        brands.add("Beer 1");
        brands.add("Beer 2");
    } else {
        brands.add("Beer 3");
        brands.add("Beer 4");
    }
    return brands;
}
```

Call the custom Java class from the activity

```
TextView brands = (TextView) findViewById(R.id.brands);
Spinner color = (Spinner) findViewById(R.id.color);

List<String> brandsList = expert.getBrands(color.getSelectedItem().toString());
StringBuilder brandsFormatted = new StringBuilder();
for (String brand : brandsList)
    brandsFormatted.append(brand).append("\n");
brands.setText(brandsFormatted);
```

References

Chapter 2 of the textbook:

Head First Android Development, 2nd Edition; by Dawn Griffiths, David Griffiths; publisher(s): O'Reilly Media, Inc.; ISBN · 9781491974056