The State Capitals App

John P. Baugh, Ph.D.

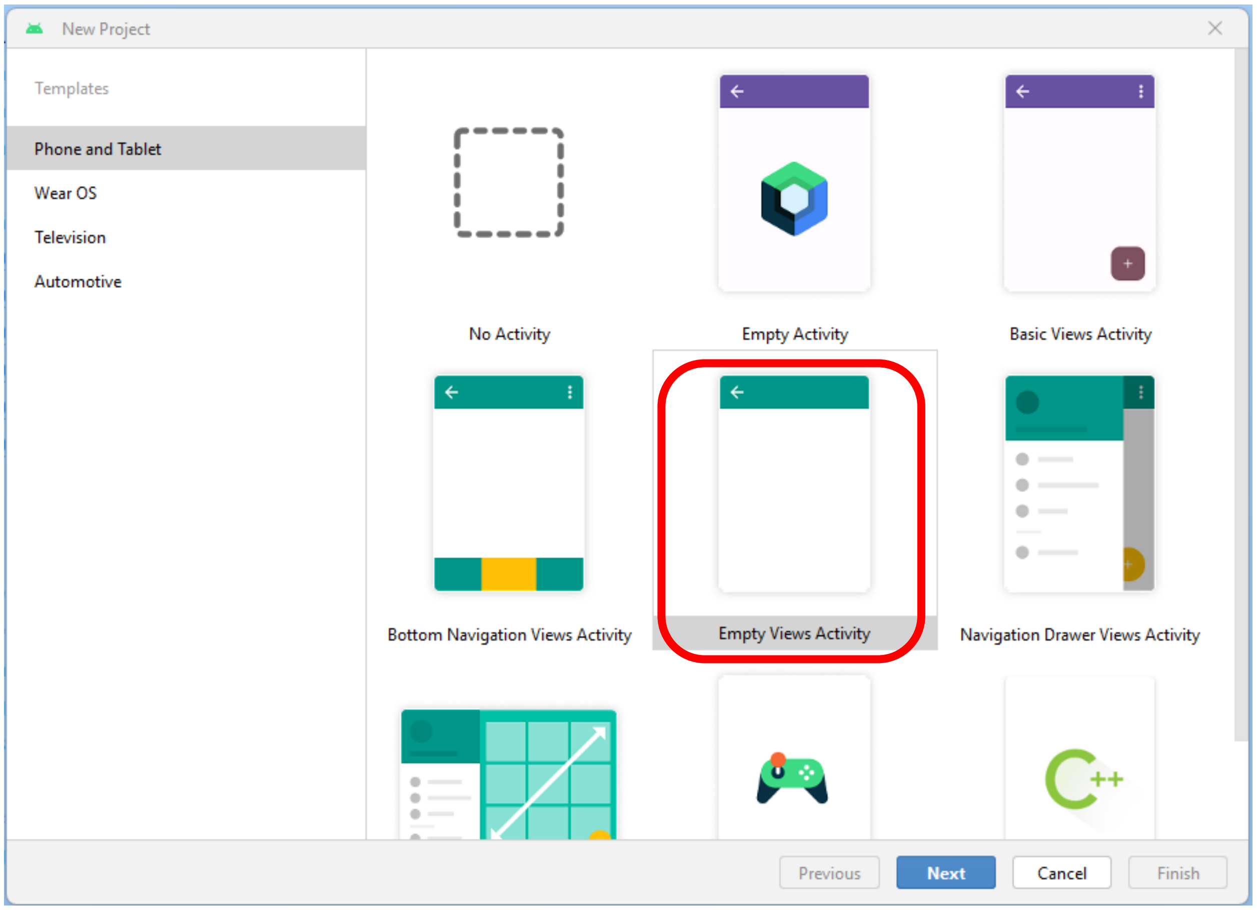
Android Development with Kotlin

Version	Date	Notes	Version Information
1	9/22/2023	Initial version	Tested in Android Studio Giraffe (2022.3.1)

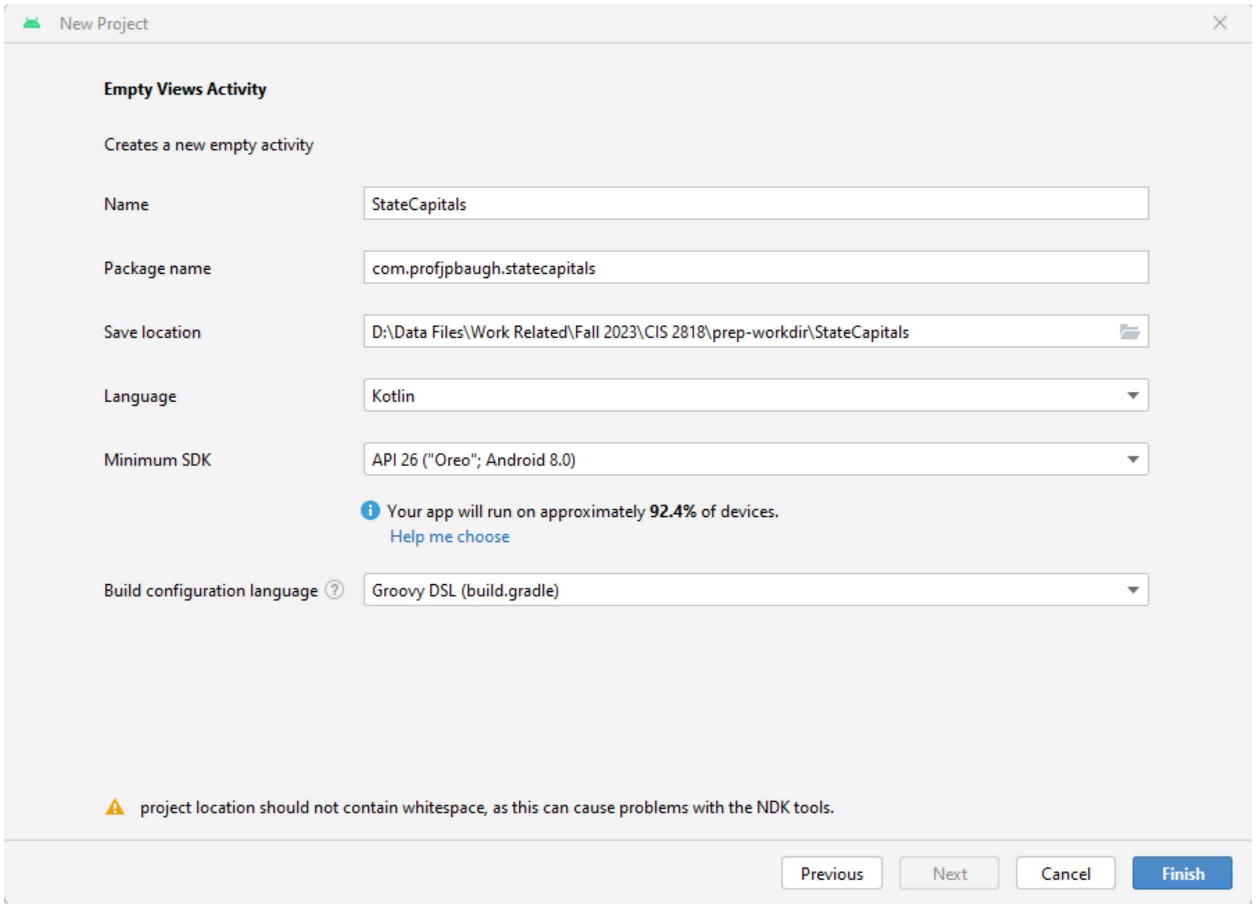
Table of Contents

Creating the Project

1. Create a new Empty Project by going to **File → New Project**
2. Select the **Empty Views Activity** project template



3. Hit the **Next** button
4. Set the settings to the following:
    a. Name: **StateCapitals**
    b. Package name: something useful, like com.yourname.statecapitals
    c. Save location: set this to somewhere you want to keep your projects
    d. Language: **Kotlin**
    e. Minimum SDK: I used **API 26 (Oreo; Android 8.0)**
    f. Build configuration language: I prefer **Groovy DSL** personally, so I switched it, but you can use Kotlin – the gradle configuration file syntax will be a bit different, though



Empty Views Activity

Creates a new empty activity

Name	StateCapitals
Package name	com.profjpbaugh.statecapitals
Save location	D:\Data Files\Work Related\Fall 2023\CIS 2818\prep-workdir\StateCapitals
Language	Kotlin
Minimum SDK	API 26 ("Oreo"; Android 8.0)
Build configuration language	Groovy DSL (build.gradle)

Your app will run on approximately **92.4%** of devices.
Help me choose

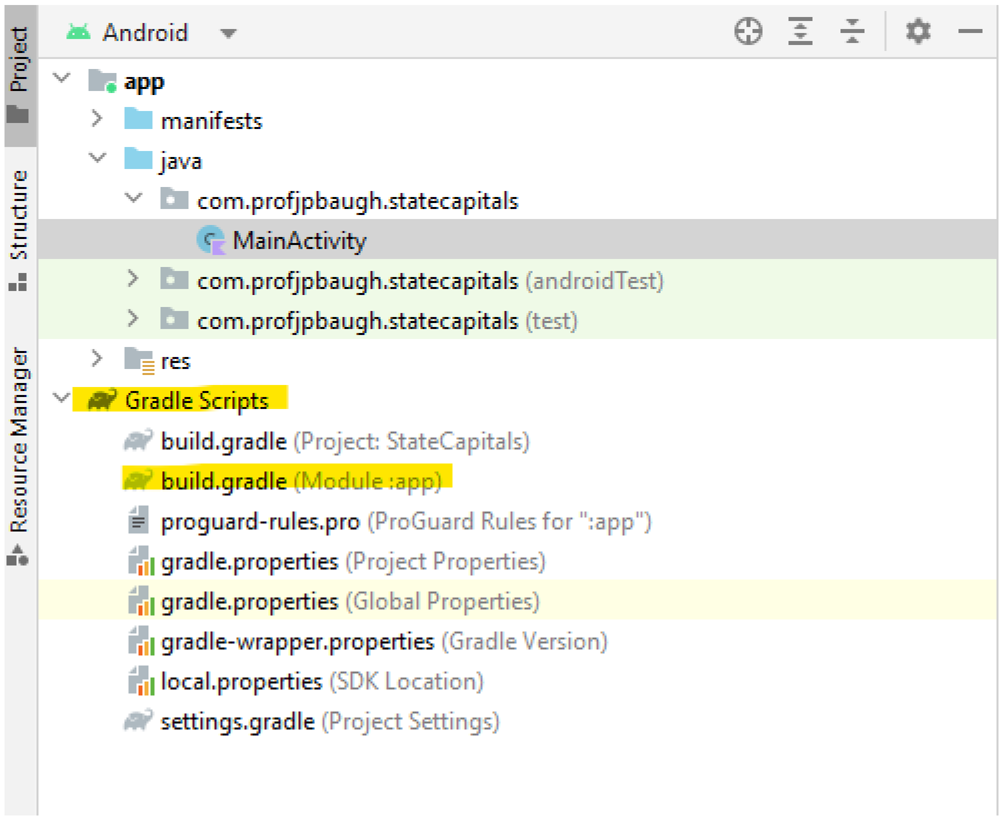
project location should not contain whitespace, as this can cause problems with the NDK tools.

5. Click **Finish**

Using View Binding

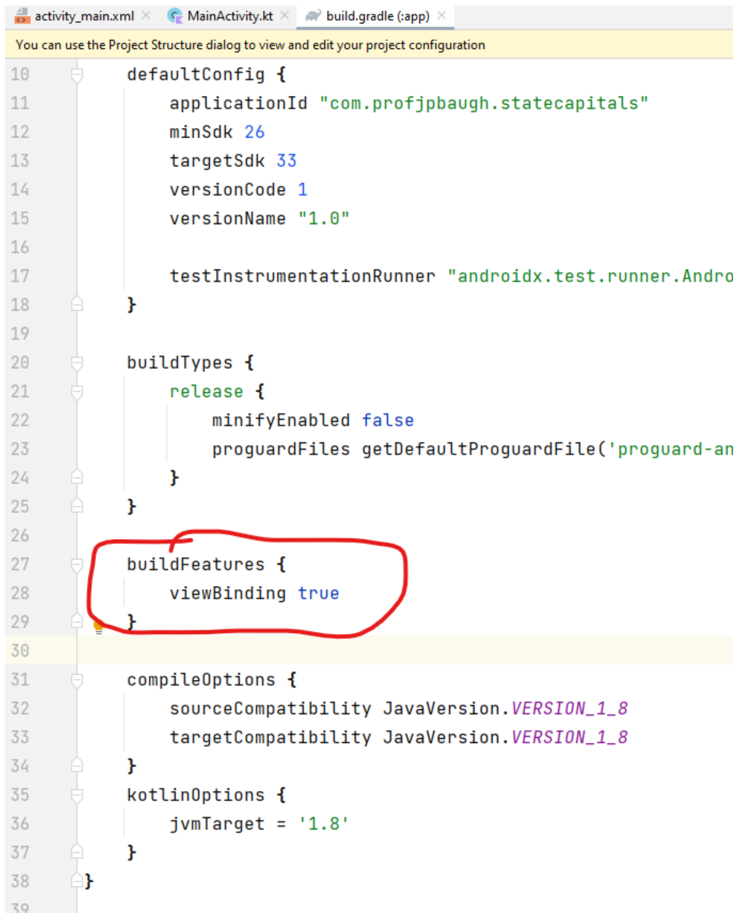
In the early days of Android development, we often used **findViewById** to obtain references to the View objects in order to manipulate them in the user interface. Later on, **Synthetics** were used after Kotlin became available. But, the newest and best technique to use now is **View Binding**. But, in order to make it work, we have to make some changes to Gradle and our code.

1. Open the Module-level **building.gradle** file under **Gradle Scripts**


Project
Structure
Resource Manager
Android
app
manifests
java
com.profjpbaugh.statecapitals
MainActivity
com.profjpbaugh.statecapitals (androidTest)
com.profjpbaugh.statecapitals (test)
res
Gradle Scripts
build.gradle (Project: StateCapitals)
build.gradle (Module :app)
proguard-rules.pro (ProGuard Rules for ":app")
gradle.properties (Project Properties)
gradle.properties (Global Properties)
gradle-wrapper.properties (Gradle Version)
local.properties (SDK Location)
settings.gradle (Project Settings)

2. Inside the **android** section, add a new sub-section **buildFeatures** as follows:

```
buildFeatures {
    viewBinding true
}
```


activity_main.xml
MainActivity.kt
build.gradle (:app)
You can use the Project Structure dialog to view and edit your project configuration

```
    defaultConfig {
        applicationId "com.profjpbaugh.statecapitals"
        minSdk 26
        targetSdk 33
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.Andro
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-an
        }
    }

    buildFeatures {
        viewBinding true
    }

    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }
}
```

3. Click **Sync Now** in the upper right corner
4. Now, modify the **MainActivity.kt** file so that the activity will now use view binding

```
package com.profjpbaugh.statecapitals

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import com.profjpbaugh.statecapitals.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {

    private lateinit var binding : ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
    }
}
```

That **binding** variable will be used later on to access various Views (user interface components, like buttons or text views/labels).

The State Capitals App

Now that we have the boilerplate code and configuration set up, we can work on the actual application, the **State Capitals App**. For this app, the user will make guesses about the capitals of states.

We need a few essential building blocks for this application:

- Capital class
    - We could implement this fully with properties

- Primary UI
    - This will be the user interface with which the user interacts
    - It should display a state name and the capital city, each in TextViews
    - There should be a button that says "Get Random State" on it

The app will display a state, and the capital city, and a button underneath. When the button ("Get Random State") is clicked, a new state and capital city will be shown. The selection of this next state and city to display should be randomized.

The data for the states and capitals should exist in the Strings Resource file (strings.xml, which resides in res -> values -> strings.xml). They can be loaded into memory and then placed appropriately in instances of the Capital class.

The Capital Class

The first thing we'll do is create an **Capital** class. This is going to be a **POKO** (**Plain Old Kotlin Class)**, in other words, no special Android-specific magic or anything particularly special about the class itself. It could even potentially be used in a different application altogether.

1. Right click the package and create a new Kotlin class
    a. Name the class **Capital**
2. Add the following code

```
package com.profjpbaugh.statecapitals

//class header, including primary constructor
class Capital(state : String, capitalCity : String) {
    //properties
    var state : String = ""
        get() = field
        set(value) {
            field = value
        }

    var capitalCity : String = ""
        get() = field
        set(value) {
            field = value
        }

    //set the property values to the corresponding
    //parameters of the primary constructor
    init {
        this.state = state
        this.capitalCity = capitalCity
    }
}
```

Capitals.kt

There are other ways we could have designed this class, including as a **data class**.

The String Array Resource

To prevent you from having to re-type everything, you could just copy and paste the string array into the strings.xml file, inside the <resources> tag.

1. Open the strings.xml file
2. Put the following string-array inside the resources tag:

```
<string-array name="states">
    <item>Alabama,Montgomery</item>
    <item>Alaska,Juneau</item>
    <item>Arizona,Phoenix</item>
    <item>Arkansas,Little Rock</item>
    <item>California,Sacramento</item>
    <item>Colorado,Denver</item>
    <item>Connecticut,Hartford</item>
    <item>Delaware,Dover</item>
    <item>Hawaii,Honolulu</item>
    <item>Florida,Tallahassee</item>
    <item>Georgia,Atlanta</item>
    <item>Idaho,Boise</item>
    <item>Illinois,Springfield</item>
    <item>Indiana,Indianapolis</item>
    <item>Iowa,Des Moines</item>
    <item>Kansas,Topeka</item>
    <item>Kentucky,Frankfort</item>
    <item>Louisiana,Baton Rouge</item>
    <item>Maine,Augusta</item>
    <item>Maryland,Annapolis</item>
    <item>Massachusetts,Boston</item>
    <item>Michigan,Lansing</item>
    <item>Minnesota,St. Paul</item>
    <item>Mississippi,Jackson</item>
    <item>Missouri,Jefferson City</item>
    <item>Montana,Helena</item>
    <item>Nebraska,Lincoln</item>
    <item>Nevada,Carson City</item>
    <item>New Hampshire,Concord</item>
    <item>New Jersey,Trenton</item>
    <item>New Mexico,Santa Fe</item>
    <item>North Carolina,Raleigh</item>
    <item>North Dakota,Bismarck</item>
    <item>New York,Albany</item>
    <item>Ohio,Columbus</item>
    <item>Oklahoma,Oklahoma City</item>
    <item>Oregon,Salem</item>
    <item>Pennsylvania,Harrisburg</item>
    <item>Rhode Island,Providence</item>
    <item>South Carolina,Columbia</item>
    <item>South Dakota,Pierre</item>
    <item>Tennessee,Nashville</item>
```

```
     <item>Texas,Austin</item>
     <item>Utah,Salt Lake City</item>
     <item>Vermont,Montpelier</item>
     <item>Virginia,Richmond</item>
     <item>Washington,Olympia</item>
     <item>West Virginia,Charleston</item>
     <item>Wisconsin,Madison</item>
     <item>Wyoming,Cheyenne</item>
</string-array>
```

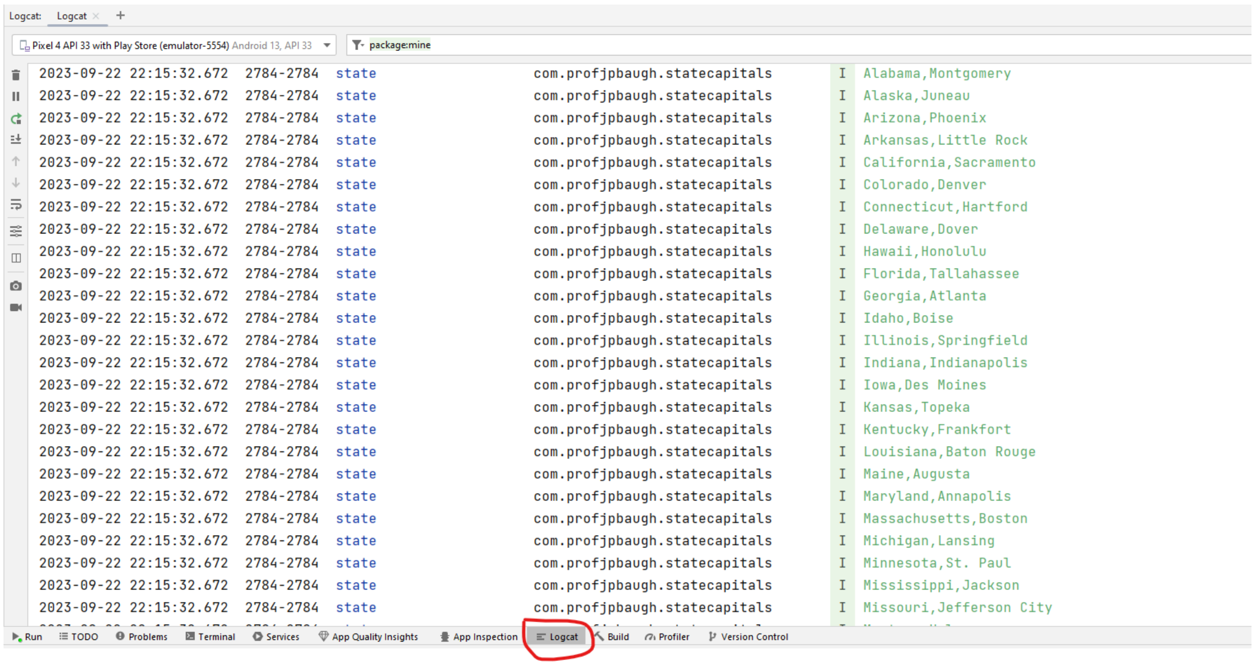
A **string array resource** is an element in a strings.xml resource file that allows you to have multiple elements, all referenced by the same name (in our case, "states"). We can use that name to pull the data into memory and store it in a data structure, with which we can then parse out the state name and the capital city name. Once we've done that, we can store them properly as instances of the Capital class.

To make sure this resource is properly available in your projectly, do the following:

3. Load the resource raw data and loop through it, printing it to Logcat, with the following code in **MainActivity.kt:**

```
package com.profjpbaugh.statecapitals

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import com.profjpbaugh.statecapitals.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {

    private lateinit var binding : ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        //testing
        val rawDataArray = resources.getStringArray(R.array.states)

        for(state in rawDataArray) {
            Log.i("state", state.toString())
        }
    }
}
```

4. Go to the Logcat tab at the bottom of Android Studio, and observe the output



```
2023-09-22 22:15:32.672  2784-2784  state                   com.profjpbaugh.statecapitals        I  Alabama,Montgomery
2023-09-22 22:15:32.672  2784-2784  state                   com.profjpbaugh.statecapitals        I  Alaska,Juneau
2023-09-22 22:15:32.672  2784-2784  state                   com.profjpbaugh.statecapitals        I  Arizona,Phoenix
2023-09-22 22:15:32.672  2784-2784  state                   com.profjpbaugh.statecapitals        I  Arkansas,Little Rock
2023-09-22 22:15:32.672  2784-2784  state                   com.profjpbaugh.statecapitals        I  California,Sacramento
2023-09-22 22:15:32.672  2784-2784  state                   com.profjpbaugh.statecapitals        I  Colorado,Denver
2023-09-22 22:15:32.672  2784-2784  state                   com.profjpbaugh.statecapitals        I  Connecticut,Hartford
2023-09-22 22:15:32.672  2784-2784  state                   com.profjpbaugh.statecapitals        I  Delaware,Dover
2023-09-22 22:15:32.672  2784-2784  state                   com.profjpbaugh.statecapitals        I  Hawaii,Honolulu
2023-09-22 22:15:32.672  2784-2784  state                   com.profjpbaugh.statecapitals        I  Florida,Tallahassee
2023-09-22 22:15:32.672  2784-2784  state                   com.profjpbaugh.statecapitals        I  Georgia,Atlanta
2023-09-22 22:15:32.672  2784-2784  state                   com.profjpbaugh.statecapitals        I  Idaho,Boise
2023-09-22 22:15:32.672  2784-2784  state                   com.profjpbaugh.statecapitals        I  Illinois,Springfield
2023-09-22 22:15:32.672  2784-2784  state                   com.profjpbaugh.statecapitals        I  Indiana,Indianapolis
2023-09-22 22:15:32.672  2784-2784  state                   com.profjpbaugh.statecapitals        I  Iowa,Des Moines
2023-09-22 22:15:32.672  2784-2784  state                   com.profjpbaugh.statecapitals        I  Kansas,Topeka
2023-09-22 22:15:32.672  2784-2784  state                   com.profjpbaugh.statecapitals        I  Kentucky,Frankfort
2023-09-22 22:15:32.672  2784-2784  state                   com.profjpbaugh.statecapitals        I  Louisiana,Baton Rouge
2023-09-22 22:15:32.672  2784-2784  state                   com.profjpbaugh.statecapitals        I  Maine,Augusta
2023-09-22 22:15:32.672  2784-2784  state                   com.profjpbaugh.statecapitals        I  Maryland,Annapolis
2023-09-22 22:15:32.672  2784-2784  state                   com.profjpbaugh.statecapitals        I  Massachusetts,Boston
2023-09-22 22:15:32.672  2784-2784  state                   com.profjpbaugh.statecapitals        I  Michigan,Lansing
2023-09-22 22:15:32.672  2784-2784  state                   com.profjpbaugh.statecapitals        I  Minnesota,St. Paul
2023-09-22 22:15:32.672  2784-2784  state                   com.profjpbaugh.statecapitals        I  Mississippi,Jackson
2023-09-22 22:15:32.672  2784-2784  state                   com.profjpbaugh.statecapitals        I  Missouri,Jefferson City
```

If you see the states and their corresponding capital, you've done everything correctly thus far. If not, check to make sure you put the string-array in the appropriate file (strings.xml)

Now let's create Capital objects for each of the pieces of data we've read in, and store them in an ArrayList for use later on.

5. Add a new field to the top, which will hold the capital/state objects later on

```kotlin
class MainActivity : AppCompatActivity() {

    private lateinit var binding : ActivityMainBinding
    private lateinit var stateList : ArrayList<Capital>
    //…
```

6. Modify the onCreate function

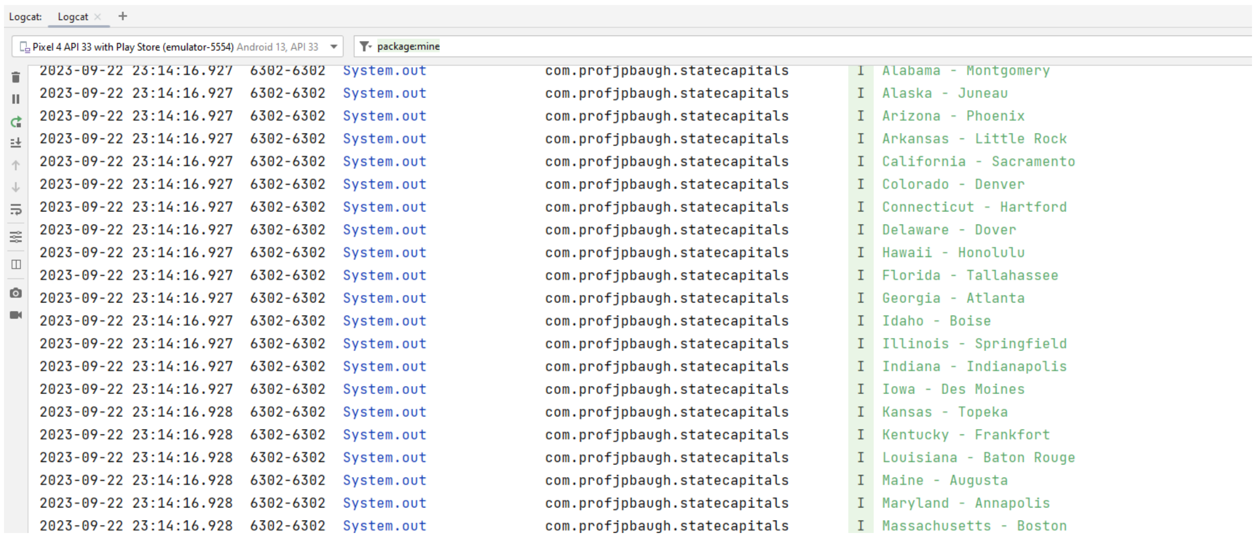
```kotlin
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    binding = ActivityMainBinding.inflate(layoutInflater)
    setContentView(binding.root)

    //testing
    val rawDataArray = resources.getStringArray(R.array.states)
    var stateName : String
    var capitolName : String
    var stringArray : List<String>
    var capital : Capital
    stateList = ArrayList<Capital>()


    // you could also look into doing this with the
    // functional style forEach function
    for(state in rawDataArray) {
        //split the strings by comma
        stringArray = state.split(",")
        stateName = stringArray[0]
        capitolName = stringArray[1]

        //create instance of Capital
        capital = Capital(stateName, capitolName)
        stateList.add(capital)
    }//end for

    //let's try the forEach style and print out the data
    // "it" represents an individual object, as the array list
    //iterates through its elements using the forEach function

    stateList.forEach {
        println("${it.state} - ${it.capitalCity}")
    }
}
```

As a reminder, the **${ }** syntax, such as in ${it.state} allows you to apply **string interpolation**. So instead of having to open and close the string literal, you just write it directly between the double quotes. This is cleaner than the alternative, which would be how you might do it in Java:

println(it.state + " – " + it.capitalCity)

That isn't *too* bad, but if you had a lot more text besides just the hyphen (-), this could be much uglier.

7. The output in Logcat should verify that the strings have been split properly, and stored in the Capital objects within the ArrayList


Logcat: Logcat

Pixel 4 API 33 with Play Store (emulator-5554) Android 13, API 33

package:mine

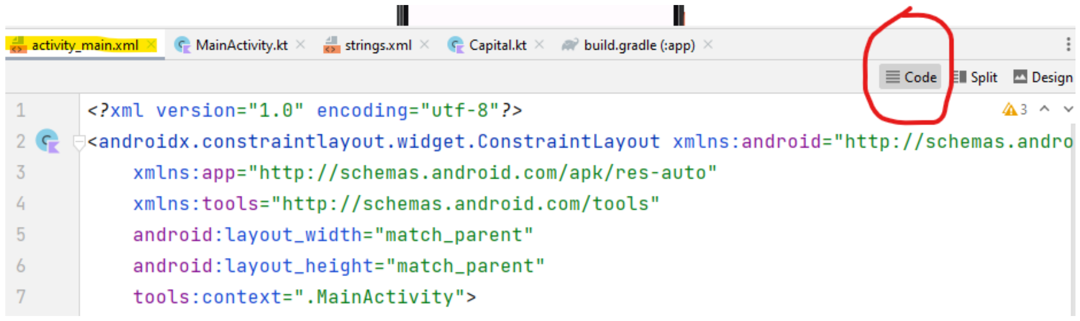
```
2023-09-22 23:14:16.927  6302-6302  System.out              com.profjpbaugh.statecapitals        I  Alabama - Montgomery
2023-09-22 23:14:16.927  6302-6302  System.out              com.profjpbaugh.statecapitals        I  Alaska - Juneau
2023-09-22 23:14:16.927  6302-6302  System.out              com.profjpbaugh.statecapitals        I  Arizona - Phoenix
2023-09-22 23:14:16.927  6302-6302  System.out              com.profjpbaugh.statecapitals        I  Arkansas - Little Rock
2023-09-22 23:14:16.927  6302-6302  System.out              com.profjpbaugh.statecapitals        I  California - Sacramento
2023-09-22 23:14:16.927  6302-6302  System.out              com.profjpbaugh.statecapitals        I  Colorado - Denver
2023-09-22 23:14:16.927  6302-6302  System.out              com.profjpbaugh.statecapitals        I  Connecticut - Hartford
2023-09-22 23:14:16.927  6302-6302  System.out              com.profjpbaugh.statecapitals        I  Delaware - Dover
2023-09-22 23:14:16.927  6302-6302  System.out              com.profjpbaugh.statecapitals        I  Hawaii - Honolulu
2023-09-22 23:14:16.927  6302-6302  System.out              com.profjpbaugh.statecapitals        I  Florida - Tallahassee
2023-09-22 23:14:16.927  6302-6302  System.out              com.profjpbaugh.statecapitals        I  Georgia - Atlanta
2023-09-22 23:14:16.927  6302-6302  System.out              com.profjpbaugh.statecapitals        I  Idaho - Boise
2023-09-22 23:14:16.927  6302-6302  System.out              com.profjpbaugh.statecapitals        I  Illinois - Springfield
2023-09-22 23:14:16.927  6302-6302  System.out              com.profjpbaugh.statecapitals        I  Indiana - Indianapolis
2023-09-22 23:14:16.927  6302-6302  System.out              com.profjpbaugh.statecapitals        I  Iowa - Des Moines
2023-09-22 23:14:16.928  6302-6302  System.out              com.profjpbaugh.statecapitals        I  Kansas - Topeka
2023-09-22 23:14:16.928  6302-6302  System.out              com.profjpbaugh.statecapitals        I  Kentucky - Frankfort
2023-09-22 23:14:16.928  6302-6302  System.out              com.profjpbaugh.statecapitals        I  Louisiana - Baton Rouge
2023-09-22 23:14:16.928  6302-6302  System.out              com.profjpbaugh.statecapitals        I  Maine - Augusta
2023-09-22 23:14:16.928  6302-6302  System.out              com.profjpbaugh.statecapitals        I  Maryland - Annapolis
2023-09-22 23:14:16.928  6302-6302  System.out              com.profjpbaugh.statecapitals        I  Massachusetts - Boston
```

The activity_main.xml Layout

In the layout file, we're going to be replacing all of the code and typing much of the XML for practice. You could certainly use the Design tools as well, but becoming familiar with some of the elements and attributes is very important and can improve your efficiency and effectiveness as a developer for Android.

In most modern software, we would

1. Open **activity_main.xml**
    a. It's under res→layout folder in the Android project structure
2. Select the **Code** tab near the upper right corner of the activity_main.xml in the window



```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.andro
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
```

3. Note that there is (probably) already code for a ConstraintLayout
4. Delete the TextView that exists there currently
    a. This should leave you with something like the following:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

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

5. Now, add a **Button** by typing in between the ConstraintLayout tags:

```
<Button
    android:id="@+id/next_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Get Next"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

The `layout_width` and `layout_height` are set to “wrap_content”, which means that they will take up as little space as possible – only what they need to hold the content and basic features of the button (default padding around the text itself, for example).

Note the `layout_constraint`***X*** attributes (where ***X*** is a direction indicator). For this button, it is being centered within the parent container (the constraint layout itself) by setting all the constraints to “parent”.

6. Now, add a **TextView**
   a. We want it to appear above the button
   b. I placed the code above the button too, but technically if you set all the constraints correctly, it doesn’t matter where in the code it is (as long as it’s inside the constraint layout)

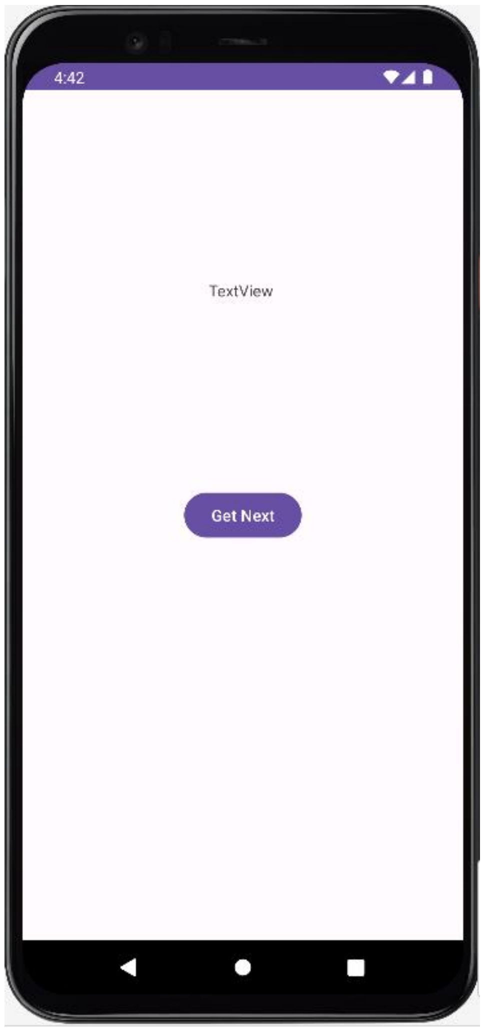
```
<TextView
    android:id="@+id/capital_info"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="TextView"
    app:layout_constraintBottom_toTopOf="@+id/next_button"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

In this case, all of the layout_constraint***X*** are set to “parent” except the layout_constraintBottom_toTopOf, which is set to the button

7. Verify that the code looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/capital_info"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="TextView"
        app:layout_constraintBottom_toTopOf="@+id/next_button"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/next_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Get Next"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

If you look at it in the **Design** tab of activity_main.xml, it should look something like the following:



Updating the code to display the states and their capitals

1. Open **MainActivity.kt**
2. **Remove** the foreach code inside onCreate, as we don't need it to display data to Logcat

```
//let's try the forEach style
~~stateList.forEach {~~
~~    println("${it.state} - ${it.capitalCity}")~~
~~}~~
```

3. Add new code to set the default state and capital information upon launch of the app

```

//Other code in onCreate goes up here…
    for(state in rawDataArray) {
        //split the strings by comma
        stringArray = state.split(",")
        stateName = stringArray[0]
        capitolName = stringArray[1]

        //create instance of Capital
        capital = Capital(stateName, capitolName)
        stateList.add(capital)
    }//end for

    //set the default information for state and capital
    //upon launch
    var capitalObject = stateList.get(Random.nextInt(stateList.size))
    var messageString = "${capitalObject.capitalCity} " +
            "is the capital of ${capitalObject.state}"
    binding.capitalInfo.setText(messageString)
}
```

4. Run the app and verify that it should information about a state and its capital city
5. Close the app and run it again and verify that it shows a state and its capital city
    a. It's very likely to be a different state and city
6. Now, let's add the code for the button

```
   //set the default information for state and capital
    //upon launch
    var capitalObject = stateList.get(Random.nextInt(stateList.size))
    var messageString = "${capitalObject.capitalCity} " +
            "is the capital of ${capitalObject.state}"
    binding.capitalInfo.setText(messageString)

    //button code
    binding.nextButton.setOnClickListener {
        capitalObject = stateList.get(Random.nextInt(stateList.size))
        messageString = "${capitalObject.capitalCity} " +
                    "is the capital of ${capitalObject.state}"
          binding.capitalInfo.setText(messageString)
     }//end setOnClickListener
}
```

Note that this code is almost identical to the code for setting the default information. We're just reusing the `capitalObject` and `messageString` variables to hold the new data each time the button is clicked.

Also note that the `Random.nextInt` function can take a parameter, which is labeled as ***until***. This means this value is excluded, but the random number generator generates values up to that value (just not including it). So, `Random.nextInt(stateList.size)` generates values from 0 up to the one less than the size of the `stateList`. So the value returned by `Random.nextInt` and passed into `stateList.get` acts as a randomized index so that a random state and capital is retrieved from the Capital objects stored in the `stateList`.

7. Run the app and make sure it randomly displays different states and their capitals as you click the button