Mobile Application Lab 1

Instructors: Legand Burge, Asad Ullah Naweed
Date: 08/23/2018
Topics: Layouts and Callbacks

Initial Setup

1. Start Android Studio and create a new project.
    a. Be sure to create a project for a phone/tablet running at least Android Oreo.
2. Delete the androidTest and test directories from your project.
3. Open the AVD Manager, and launch a new Android Virtual Device running Android Oreo.
4. Build and run your newly created project on the Virtual Device.

Layout Modifications

Open the `activity_main.xml` in the `src/main/res` directory of your project. Be sure to click on the "Text" tab to view the XML. You should see the following:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.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="lab1.techexchange.com.lab1.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"/>

</android.support.constraint.ConstraintLayout>
```

This is XML code that defines your layout. An XML file defines a tree-like data structure. In this particular example, the root is the ConstraintLayout, with a single TextView as its child. The ConstraintLayout acts as a container for the views inside it.

First, add an ID to the TextView as follows:

```xml
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.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="lab1.techexchange.com.lab1.MainActivity">

  <TextView
      android:id="@+id/app_text_view"
      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"/>

</android.support.constraint.ConstraintLayout>
```

This ID will allow you to reference this TextView from other views, as we'll see ahead. Make the following changes to your XML file:

```xml
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.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="lab1.techexchange.com.lab1.MainActivity">

  <TextView
      android:id="@+id/app_text_view"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Do chocolate cupcakes contain chocolate?"
      app:layout_constraintBottom_toTopOf="@id/yes_button"
```

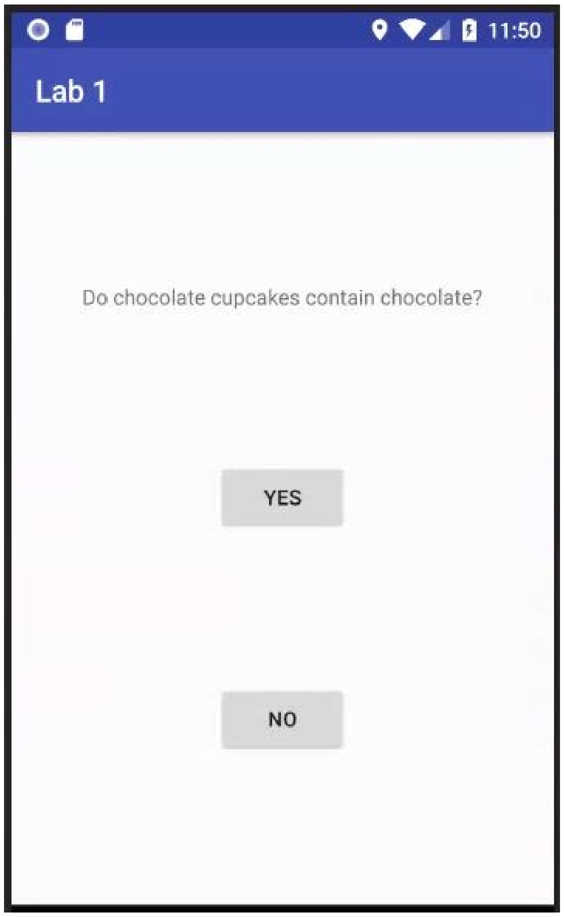
```
       app:layout_constraintLeft_toLeftOf="parent"
       app:layout_constraintRight_toRightOf="parent"
       app:layout_constraintTop_toTopOf="parent" />

   <Button
       android:id="@+id/yes_button"
       android:layout_height="wrap_content"
       android:layout_width="wrap_content"
       android:text="Yes"
       app:layout_constraintTop_toBottomOf="@id/app_text_view"
       app:layout_constraintLeft_toLeftOf="parent"
       app:layout_constraintRight_toRightOf="parent"
       app:layout_constraintBottom_toTopOf="@id/no_button"
       />

   <Button
       android:id="@+id/no_button"
       android:layout_height="wrap_content"
       android:layout_width="wrap_content"
       android:text="No"
       app:layout_constraintTop_toBottomOf="@id/yes_button"
       app:layout_constraintLeft_toLeftOf="parent"
       app:layout_constraintRight_toRightOf="parent"
       app:layout_constraintBottom_toBottomOf="parent"
       />

</android.support.constraint.ConstraintLayout>
```

Before proceeding further, make sure that your app builds and runs on your emulator successfully. You should see something like this:



Now, read the XML that went into creating these buttons. It basically describes how the individual components are laid out with respect to each other, and they reference each other by their IDs. These IDs are also a way for us to get handles to these objects in Java code, as we'll see later during the lab.

You should be able to tap on the buttons, though they'll not do anything right now.

Actions and Listeners

We'll now attach "listeners" to each of the buttons. A listener is simply a method that gets called when the button is pressed.

Before proceeding further, search for the `build.gradle` file in the `app` directory. Note that there are two files with this name but in different directories, so be sure to find the correct one.

The file should contain a block of code named `android`. Add the following lines inside the block:

```
android {
    compileSdkVersion 26
    defaultConfig {
        applicationId "lab1.techexchange.com.lab1"
        minSdkVersion 26
        targetSdkVersion 26
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
}
```

These lines will enable you to use Java 8 language features for much more concise code. Most notably, it will allow you to write your listeners using lambdas instead of the much more verbose anonymous classes.

Now open up the `MainActivity.java` file in your project. It should be somewhere in the `src/main/java` directory of your app project. There should be a single class in there called `MainActivity`. It should look something like this:

```
public class MainActivity extends AppCompatActivity {

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

We will now define some behaviors that will get executed when the buttons are pressed. Add the following bits of code to the `onCreate` method:

```java
public class MainActivity extends AppCompatActivity {

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

    Button yesButton = findViewById(R.id.yes_button);
    yesButton.setOnClickListener(view -> {
      Toast.makeText(MainActivity.this, "Correct!", Toast.LENGTH_SHORT).show();
    });

    Button noButton = findViewById(R.id.no_button);
    noButton.setOnClickListener(view -> {
      Toast.makeText(MainActivity.this, "Wrong!", Toast.LENGTH_SHORT).show();
    });
  }
}
```

Now, try and run your app again. Notice what happens when you tap on the buttons. You will see Toast messages when you press the buttons. Toast messages provide a very simple way to show a short message in the Activity's UI, and don't need to be defined in the app's layout because they are not permanent.