CP3406_CP5307 Codelab 4.1: Create a fragment

Contents

1. App overview

In the three codelabs that make up this week's pracs, you will work on an app called AndroidTrivia. The completed app is a game in which the user answers three trivia questions about Android coding. If the user answers all three questions correctly, they win the game and can share their results.

The AndroidTrivia app illustrates navigation patterns and controls. The app has several components:

- In the title screen, the user starts the game.
- In the game screen with questions, the user plays the game and submits their answers.
- The navigation drawer slides out from the side of the app and contains a menu with a header. A drawer icon opens the navigation drawer. The navigation-drawer menu contains a link to the About page and a link to the rules of the game.

The top of the app displays a view called the *app bar* (or *action bar*) which shows the app's name.

2. Explore the starter app project

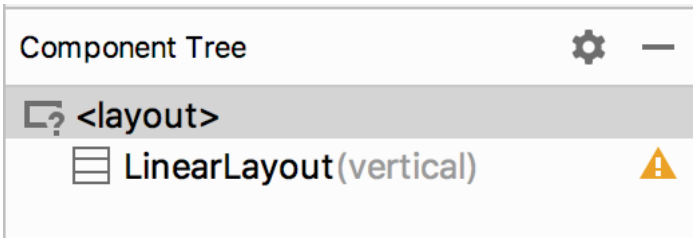
In this codelab, you work from a starter app that provides template code and Fragment classes that you need as you complete the Trivia app.

1. Download the starter code from LearnJCU (*cp3406_cp5307-4.1-starter.zip*), which is provided alongside this week's prac sheet.

2. Unzip the folder, and then import/open the project in Android Studio and run the app. When the app opens, it doesn't do anything other than display the app name and a blank screen.

3. In the Android Studio Project pane, open the Project: Android view to explore the project files. Open the **app > kotlin+java** folder to see the `MainActivity` class and Fragment classes.

4. Open the **res > layout** folder and double-click **activity_main.xml**. The `activity_main.xml` file appears in the Layout Editor.
5. Open the **Design** tab. The **Component Tree** for the `activity_main.xml` file shows the root layout as vertical `LinearLayout`.



In a vertical linear layout, all the child views in the layout are aligned vertically.

3. Add a fragment

A `Fragment` represents a behavior or a portion of user interface (UI) in an `Activity`. You can combine multiple fragments in a single activity to build a multi-pane UI, and you can reuse a `Fragment` in multiple activities.

Think of a `Fragment` as a modular section of an activity, something like a "sub-activity" that you can also use in other activities:

- A `Fragment` has its own lifecycle and receives its own input events.
- You can add or remove a `Fragment` while the activity is running.
- A `Fragment` is defined in a Kotlin class.
- A `Fragment`'s UI is defined in an XML layout file.

The AndroidTrivia app has a main activity and several fragments. Most of the fragments and their layout files have been defined for you. In this task, you create a fragment and add the fragment to the app's main activity.

Step 1: Add a Fragment class

In this step, you create a blank `TitleFragment` class. Start by creating a Kotlin class for a new Fragment:

1. In Android Studio, click anywhere inside the Project pane to bring the focus back to the project files. For example, click the **com.example.androidtriviastarter** folder.

Select File > New > Fragment > Fragment (Blank).

2. For the Fragment name, enter **TitleFragment**.
3. For the Fragment layout name, enter **placeholder_layout** (we will not use this layout for our app as it already has the layout designed for TitleFragment).
4. For source language, select **Kotlin**.
5. Click **Finish**.
6. Open the `TitleFragment.kt` fragment file, if it is not already open. It contains the `onCreateView()` method, which is one of the methods that's called during a Fragment's lifecycle.
7. Delete the code inside `onCreateView()`. The `onCreateView()` function is left with only the following code:

```
override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
                          savedInstanceState: Bundle?): View? {
}
```

8. In `TitleFragment` class, delete the `onCreate()` method, the fragment initialization parameters and companion object. Make sure your `TitleFragment` class looks like the following:

```
class TitleFragment : Fragment() {

    override fun onCreateView(inflater: LayoutInflater, container:
ViewGroup?, savedInstanceState: Bundle?): View? {



    }
}
```

Create a binding object

The Fragment won't compile now. To make the Fragment compile, you need to create a binding object and inflate the Fragment's view (which is equivalent to using `setContentView()` for an Activity).

1. In the `onCreateView()` method in `TitleFragment.kt`, create a `binding` variable (`val binding`).
2. To inflate the Fragment's view, call the `DataBindingUtil.inflate()` method on the Fragment's `Binding` object, which is `FragmentTitleBinding`.

Pass four arguments into the `DataBindingUtil.inflate` method:

- `inflater`, which is the `LayoutInflater` used to inflate the binding layout.
- The XML layout resource of the layout to inflate. Use one of the layouts that is already defined for you, `R.layout.fragment_title`.
- `container` for the parent `ViewGroup`. (This parameter is optional.)
- `false` for the `attachToParent` value.

3. Assign the binding that `DataBindingUtil.inflate` returns to the `binding` variable.
4. Return `binding.root` from the method, which contains the inflated view. Your `onCreateView()` method now looks like the following code:

```
override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
                          savedInstanceState: Bundle?): View? {
   val binding = DataBindingUtil.inflate<FragmentTitleBinding>(inflater,
           R.layout.fragment_title,container,false)
   return binding.root
   }
```

5. Open **res>layout** and delete `placeholder_layout.xml`.

Step 2: Add the new fragment to the main layout file

In this step, you add the `TitleFragment` to the app's `activity_main.xml` layout file.

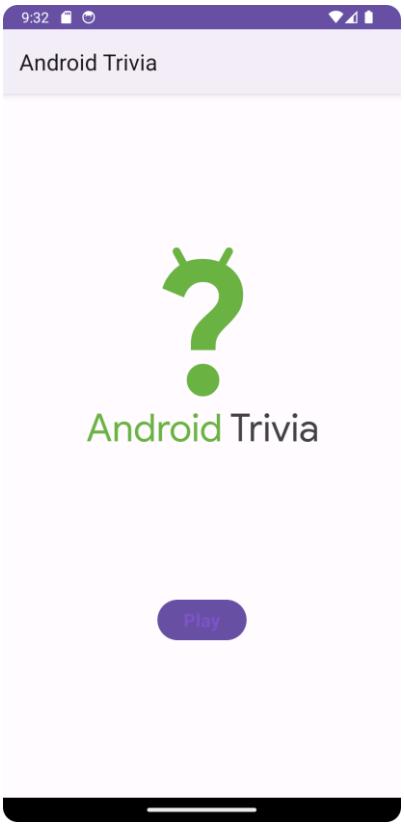
1. Open **res > layout > activity_main.xml** and select the **Code** tab to view the layout XML code.
2. Inside the existing `LinearLayout` element, add a `fragment` element.
3. Set the fragment's ID to `titleFragment`.
4. Set the fragment's name to the full path of the Fragment class, which in this case is `com.example.androidtriviastarter.TitleFragment`.
5. Set the layout width and height to `match_parent`.

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="vertical">
            <fragment
                android:id="@+id/titleFragment"
                android:name="com.example.androidtriviastarterTitleFragment"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                />
        </LinearLayout>

</layout>
```

6. Run the app. The Fragment has been added to your main screen.



4. Summary

In this codelab, you added a Fragment to the AndroidTrivia app, which you will keep working on in the next two codelabs in this lesson.

- A *Fragment* is a modular section of an activity.
- A Fragment has its own lifecycle and receives its own input events.
- Use the `<fragment>` tag to define the layout for the Fragment in the XML layout file.
- Inflate the layout for a Fragment in `onCreateView()`.
- You can add or remove a Fragment while the activity is running.