Labsheet 8 – Friday 12th November, 2021

Fragments for Larger Interfaces

Software Development for Portable Devices

Information about HD TA:

AASHITA DUTTA <h20201030130@hyderabad.bits-pilani.ac.in>

Breakout room link: https://meet.google.com/erq-uxqi-egv

Information about FD TA:

ARUNEEMA DESHMUKH <f20180568@hyderabad.bits-pilani.ac.in>

Breakout room link: https://meet.google.com/pvo-osiz-cuh

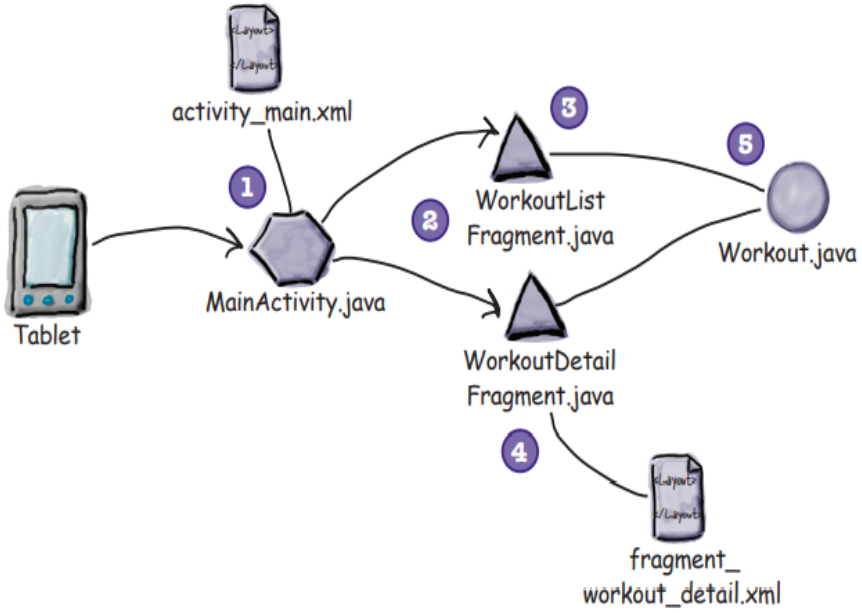
Introduction

In the last lab, we covered the workout app for the phone version. If the app's running on a phone, it displays the workout details in a separate activity. If the app's running on a tablet, we would like to display details of the workout next to the list of workouts.

Android keeps track of places the user has visited within an app by adding them to the back stack as separate transactions. By learning about back stack and fragment transactions, you'll be able to use the back button to control the program's behavior. Finally, you'll learn how to save and restore a fragment's state.

In this practical, we shall see that by having your app look and function differently based on the device it's running on, you can create flexible user interfaces. We can make apps look different on different devices by putting separate layouts in device-appropriate folders.
Tablet version of the app-

1. When the app gets launched, it starts MainActivity as before and uses activity_main.xml as layout.
2. MainActivity's layout displays two fragments, WorkoutListFragment and WorkoutDetailFragment.
3. WorkoutListFragment displays a list of workouts, without extra layout file.
4. When the user clicks on one of the workouts, its details are displayed in WorkoutDetailFragment and uses fragment_workout_detail.xml as layout file.
5. Both fragments get their workout data from Workout.java as before.



There are two key differences between the phone and tablet versions. The first is that MainActivity's layout needs to display both fragments, not just WorkoutListFragment. The second difference is that we no longer need to start DetailActivity when the user clicks on one of the workouts. Instead, we need to display WorkoutDetailFragment in MainActivity.

Try it yourself: Create a tablet AVD and create a new layout that's designed to work on devices with larger screens. You can display details of the first workout in the first instance and later update it to display details of the workout the user selects.

Task 1: Create Tablet AVD

1. Open your Workout project from lab 7 in android studio.
2. Open the Android Virtual Device Manager. You'll be presented with a screen showing you a list of the AVDs you've already set up. Click on the Create Virtual Device button at the bottom of the screen.

3. Select the hardware: On the next screen, you'll be prompted to choose a device definition, the type of device your AVD will emulate. Choose Tablet from the Category menu and Nexus 7 from the list. Then click the Next button.
4. Select a system image: The system image gives you an installed version of the Android OS. You can choose the version of Android you want to be on your AVD. For example, if you want your app to work on a minimum of API level 19, choose a system image for at least API level 19. Then click on the Next button.
5. Verify the AVD configuration: On the next screen, verify the AVD configuration. This screen summarizes the options you chose over the last few screens, and gives you the option of changing them. Change the screen startup orientation to **Landscape**, then click on the Finish button.

Task 2: Create Tablet layout

1. Put screen-specific resources in screen-specific folders: If you want to have one layout for large screen devices such as tablets, and another layout for smaller devices such as phones, you put the layout for the tablet in the **app/src/main/res/layout-large folder**, and the layout for the phone in the app/src/main/ res/layout folder.

   You can put all kinds of resources (drawables or images, layouts, menus, and values) in different folders to specify which types of device they should be used with. The screen-specific folder name can include screen size, density, orientation and aspect ratio, with each part separated by hyphens.

2. If you only want your app to work on devices with particular screen sizes, you can specify this in AndroidManifest.xml using the attribute. As an example, if you don't want your app to run on devices with small screens, you'd use: `<support-screens android:smallScreens="false"/>`
3. Layout-large Folder: Click on app/src/main/res folder, and choose File→New...→Directory. When prompted, give the folder a name of "layout-large". When you click on the OK button. copy our existing activity layout file activity_main.xml into the app/ src/main/res/layout-large folder. [If this folder is not visible in Android Studio use File Explorer]
4. Update layout to display two fragments:

   activity _main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="horizontal" >
   <fragment
       android:name=".WorkoutListFragment"
       android:id="@+id/list_frag"
       android:layout_width="0dp"
       android:layout_weight="2"
       android:layout_height="match_parent" />
   <fragment
       android:name=".WorkoutDetailFragment"
       android:id="@+id/detail_frag"
       android:layout_width="0dp"
       android:layout_weight="3"
       android:layout_height="match_parent" />
</LinearLayout>
```

Resources: https://developer.android.com/guide/practices/ screens_support.html

Questions:

1. Android decides at runtime which resources to use by checking the device's specification and looking for the best match. What will happen If there's no exact match?
2. If resources are only available for screens larger than the current one, then will the app work?

Run the app. When you run the app on a tablet, MainActivity displays a list of workout names on the left, and details of the first workout appear next to it.

Task 3: Display workout selected by user

The current code in Main Activity starts DetailActivity whenever the user clicks on one of the workouts. We need to change the code so that this only happens if the app's running on a device with a small screen such as a phone. For a large screen, the WorkoutDetailFragment should be in the MainActivity.

The WorkoutDetailFragment updates its views when it is started. But once the fragment is displayed on screen, to get the fragment to update the details, we'll replace the detail fragment with a brand-new detail fragment, each time we want its text to change. [Why?]

In every app we've built so far, when we've clicked on the **Back button**, we've been returned to the previous activity. This is standard Android behavior, and something that Android has handled for us automatically. However, if we're running this particular app on a tablet, we don't want the Back button to return us to the previous activity. We want it to return us to the **previous fragment state**.

We're going to replace the entire WorkoutDetailFragment with a new instance of it each time the user selects a different workout. Each new instance of WorkoutDetailFragment will be set up to display details of the workout the user selects. That way, we can add each fragment replacement to the back stack as a separate **transaction**.

1. Update activity_main.xml: We add a fragment in a layout using a <FrameLayout> whenever we need to replace fragments programmatically, such as adding fragment changes to the back stack. In activity_main.xml replace the second fragment with a FrameLayout.
2. Differentiate between devices in MainActivity: If we can get our MainActivity code to check for the existence of a view with an ID of fragment_container, we can get MainActivity to behave differently depending on whether the app's running on a phone or a tablet.

```
View fragmentContainer = findViewById(R.id.fragment_container);
if (fragmentContainer != null) {
    //Add the fragment to the FrameLayout
}
else {
}
```

Task 4: Fragment transactions

We **add, replace, or remove** fragments at runtime using a fragment transaction. A fragment transaction is a set of changes relating to the fragment that you want to apply, all at the same time.

When you create a fragment transaction, you need to do three things:

1. **Begin the transaction.**

This tells Android that you're starting a series of changes that you want to record in the transaction.

2. **Specify the changes.**

These are all the actions you want to group together in the transaction. This can include adding, replacing, or removing a fragment, updating its data, and adding it to the back stack.

3. **Commit the transaction.**

This finishes the transaction and applies the changes.

Try it yourself:

- Begin a transaction, specify the changes. You can use transaction.setTransition(transition); to set the animation.
- Once you've specified all the actions [here replace()] you want to take as part of the transaction, you can use the addToBackStack() method to add the transaction to the back stack. This method takes one parameter, a String name you can use to label the transaction. This parameter is needed if you need to programmatically retrieve the transaction. Most of the time you won't need to do this, so you can pass in a null value.
- Commit the transaction.

MainActivity.java

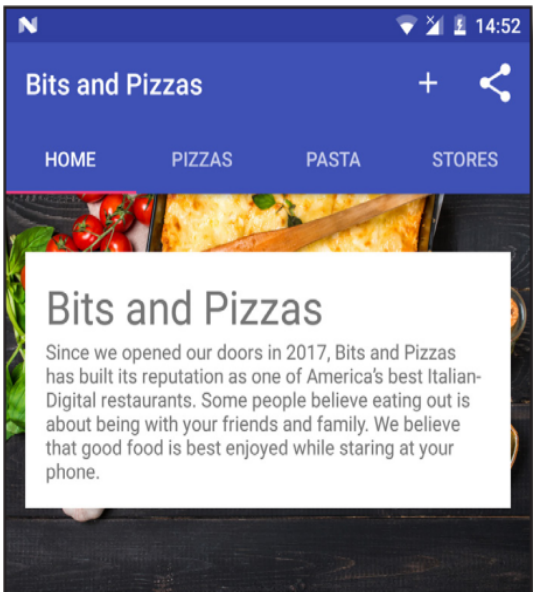
```
package com.example.workout;

import androidx.appcompat.app.AppCompatActivity;
import androidx.fragment.app.FragmentTransaction;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity implements
WorkoutListFragment.Listener {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);
   }
   @Override
   public void itemClicked(long id) {
       View fragmentContainer = findViewById(R.id.fragment_container);
       if (fragmentContainer != null) {
           WorkoutDetailFragment details = new WorkoutDetailFragment();
           FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
           details.setWorkout(id);
           ft.replace(R.id.fragment_container, details);
           ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
           ft.addToBackStack(null);
           ft.commit();
       }
       else {
```

```
            Intent intent = new Intent(this, DetailActivity.class);
            intent.putExtra(DetailActivity.EXTRA_WORKOUT_ID, (int)id);
            startActivity(intent);
        }
    }
}
```

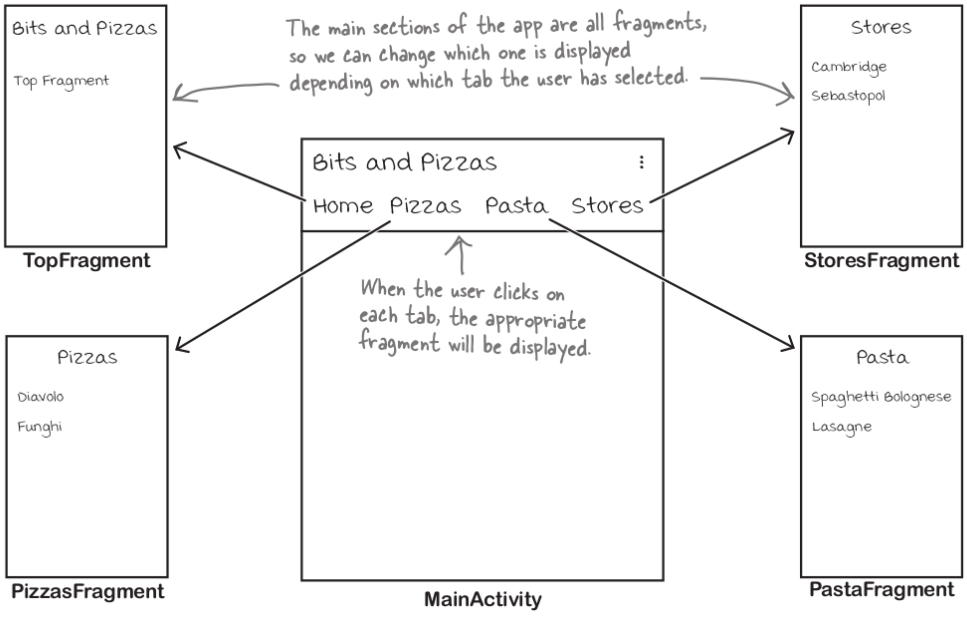
Run the app. Try rotating the screen and see the changes.

When we rotate the device, Android destroys and recreates the activity. When this happens, local variables used by the activity can get lost. To prevent this from happening, we can save the state (Saving state with fragments) of our local variables in the fragment's onSaveInstanceState() method and then use the fragment's onCreate() method to restore the state.

Exercise: Add tabs to BitsAndPizzas App



Change the Bits and Pizzas app so that it uses tab navigation. Display a set of tabs underneath the toolbar, with each option on a different tab. When the user clicks on a tab, the screen for that option will be displayed. Enable the user to swipe left and right between the different tabs.



The steps to be followed are:

- **Create the fragments:** Create basic versions of TopFragment, PizzaFragment, PastaFragment, and StoresFragment. You can use ListFragments.

- **Enable swipe navigation between the fragments**. Update MainActivity so that the user can swipe between the different fragments. To do this use a view pager, which is a view group that allows you to swipe through different pages in a layout, each page containing a separate fragment. [Slide between fragments using ViewPager]
  - ➔ To get a view pager to display a fragment on each of its pages, there are two key pieces of information you need to give it: the number of pages it should have `getCount()`, and which fragment should appear on each page `getItem(int position)`.
  - ➔ You do this by creating a fragment pager adapter, and adding it to your activity code.

```
SectionsPagerAdapter pagerAdapter =
        new SectionsPagerAdapter(getSupportFragmentManager());
ViewPager pager = (ViewPager) findViewById(R.id.pager);
pager.setAdapter(pagerAdapter);
```

- **Add the tab layout**. Add a [TabLayout] to MainActivity (inside AppBarLayout after Toolbar) that will work in conjunction with the swipe navigation. The user will then be able to navigate to each fragment by clicking on a tab, or swiping between them.
  - ➔ Use `public CharSequence getPageTitle(int position)` set the tab text.
  - ➔ To set up TabLayout with ViewPager use `tabLayout.setupWithViewPager(pager);`

Refer:
https://guides.codepath.com/android/ViewPager-with-FragmentPagerAdapter
https://guides.codepath.com/android/Google-Play-Style-Tabs-using-TabLayout