Labsheet 7 – Thursday 29th October 2021

Fragments

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>

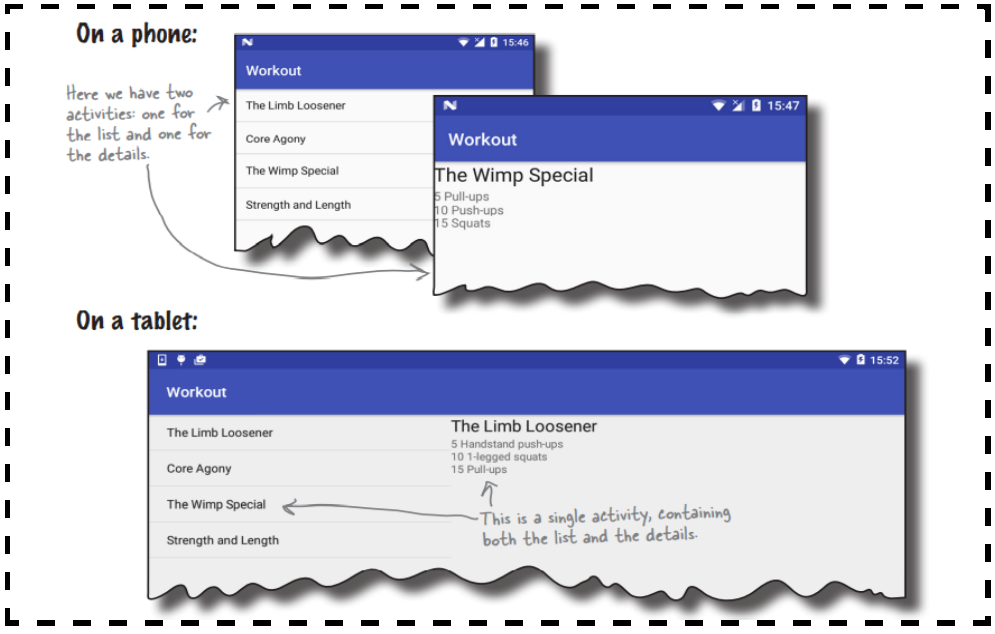
Breakout room link: https://meet.google.com/pvo-osiz-cuh

Introduction

To run the same app on different devices like phones or tablets, It's not enough to simply have different layouts for different devices. You also need different Java code to run alongside the layouts so that the app can behave differently depending on the device. **Fragments** help prevent this duplicity of code. Fragments are like reusable components or sub-activities. A fragment is used to control part of a screen, and can be reused between screens. Fragments like activity also have associated layouts.

App Description: Workout App

In this Workout app example, for instance, we need to provide one activity for tablets, and two activities for phones as shown below. But the problem is that the same code needs to be run by multiple activities.



In this practical, you learn how to create basic fragments and list fragments, how to add them to your activities, and how to get your fragments and activities to communicate with one another by building the Workout app. For now let us focus on building PhoneVersion of app. For now we're building the phone UI, but later on we'll be able to reuse the fragments to create a different UI for a tablet.

1. Create a basic app with a single activity MainActivity and layout.
    - Contains a fragment called WorkoutListFragment that displays a list of workouts.
2. On clicking one of the workouts from list, Detail Activity starts with layout as activity_detail.xml
    - Contains a fragment called WorkoutDetailFragment that displays details of workout the user selected.
3. WorkoutDetailFragment uses fragment_workout_detail.xml for its layout.
    - It displays the Title and Description of the workout the user has selected.
4. WorkoutListFragment and WorkoutDetailFragment get their workout data from Workout.java.
    - Workout.java contains an array of Workouts.

Task 1: WorkoutDetailFragment

Try it yourself: When the app is launched, MainActivity gets created. The user clicks on the button in MainActivity to start DetailActivity having activity_detail.xml as a layout file. activity_detail.xml includes an element referring to WorkoutDetailFragment having fragment_workout_detail.xml as a layout file that inflates the layout to a View object. activity_detail.xml's Views are inflated to View Java objects (WorkoutDetailFragment's View object in place of the <fragment> element in its layout's XML). Finally, DetailActivity is displayed on the device.

1. Create a new Android project with an **Empty Activity** as Project Template for an application named "Workout". You can leave the name of the activity as Main Activity.
2. Add a Button: To work on the fragment for DetailActivity, adding a button to MainActivity will give us an easy way of navigating from MainActivity to DetailActivity as we will code for it first. Let the Button have a text value of **@string/details_button.** Add a string resource in strings.xml called "details_button".Give it a value, this value is the text that will appear on the Button.
3. Add an Intent: We need to get MainActivity's button to start DetailActivity when it's clicked. To do this, we'll add a method called onShowDetails() to MainActivity. The method will start DetailActivity using an intent.
4. New fragment: Now add a new fragment called WorkoutDetailFragment to the project to display details of a single workout. To create the new fragment, choose **File → New → Fragment -> Fragment(Blank).** Name the fragment "WorkoutDetailFragment", and give the fragment layout a name of "fragment_workout_detail". Uncheck the options to include fragment factory methods and interface callbacks if present.
5. To create a fragment, you first need to extend the Fragment class. Go to WorkoutDetailFragment created in the app/ src/main/java folder and implement the onCreateView method() (**this may already be present**), which gets called each time Android needs the fragment's layout. This method returns a View object that represents the fragment's user interface. The first parameter is a LayoutInflator that you can use to inflate the fragment's layout. Inflating the layout turns your XML views into Java objects.
    - ❖ The second parameter is a ViewGroup. This is the ViewGroup in the activity's layout that will contain the fragment.

- ❖ The final parameter is a Bundle. This is used if you've previously saved the fragment's state, and want to reinstate it.
- ❖ Inside onCreateView, LayoutInflator's inflate() method specify what layout the fragment should use,i.e. R.layout.fragment_ workout_detail.

WorkoutDetailFragment.java

```
package com.example.workout;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.view.LayoutInflater;
import android.view.ViewGroup;

public class WorkoutDetailFragment extends Fragment{

   @Override
   public view onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
       return inflater.inflate(R.layout.fragment_workout_detail, container,
false);
   }
}
```

Questions:

Why must all fragments have a public no-argument constructor?

LayoutInflator's inflate() method is equivalent to which method of activity?

6. WorkoutDetailFragment Layout: Now update the layout code (fragment_workout_detail.xml) so that our fragment contains two text views, one for the workout title and one for the workout description.
7. DetailActivity Layout: Add a fragment WorkoutDetailFragment to DetailActivity activity's layout so that the fragment gets displayed in the activity's layout. https://developer.android.com/guide/fragments/create#add-xml
8. If your layout contains a single fragment, the element can be the layout file's root. When Android creates the activity's layout, it replaces the element with the View object returned by the fragment's onCreateView() method at runtime.

Questions:

What is the logic behind removing the root element in layout? What if we have multiple fragments, do we need to group them under the root element?

9. WorkoutDetailFragment Layout: Change WorkoutDetailFragment so that it displays details of a workout instead of the placeholder text (android:text)

fragment_workout_detail.xml

```
<?xml version="1.0" encoding="utf-8"?>
<<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView
        android:id="@+id/textTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge" />
    <TextView
        android:id="@+id/textDescription"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

10. Workout.java: Let's write a Java class called Workout.java that defines an array of four workouts, where each workout is composed of a name and description. The data will be used by the WorkoutDetailFragment to display details of a particular workout.

Workout.java

```
package com.example.workout;

public class Workout{
    private String name;
    private String description;
    public static final Workout[] workouts = {
        new Workout("The Limb Loosener", "5 Handstand push-ups\n10 1-legged
squats\n15 Pull-ups"),
        new Workout("Core Agony", "100 Pull-ups\n100 Push-ups\n100 Sit-ups\n100
Squats"),
        new Workout("The Wimp Special", "5 Pull-ups\n10 Push-ups\n15 Squats"),
        new Workout("Strength and Length", "500 meter run\n21 x 1.5 pood
kettleball swing\n21 x pull-ups")
    };
    private Workout(String name, String description) {
        this.name = name;
        this.description = description;
    }

    public String getDescription() {
        return description;
    }

    public String getName() {
        return name;
    }

    public String toString() {
        return this.name;
    }

}
```

11. Workout ID: WorkoutDetailFragment has to display workout details, so add a simple setter method to the fragment that sets the value of the workout ID. The activity will then be able to use this method to set the workout ID. Update the WorkoutDetailFragment code:

WorkoutDetailFragment.java

```
package com.example.workout;

import android.os.Bundle;

import androidx.fragment.app.Fragment;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

/**
 * A simple {@link Fragment} subclass.
 * Use the {@link WorkoutDetailFragment#newInstance} factory method to
 * create an instance of this fragment.
 */
public class WorkoutDetailFragment extends Fragment {

    // TODO: Rename parameter arguments, choose names that match
    // the fragment initialization parameters, e.g. ARG_ITEM_NUMBER
    private static final String ARG_PARAM1 = "param1";
    private static final String ARG_PARAM2 = "param2";

    private long workoutId;

    // TODO: Rename and change types of parameters
    private String mParam1;
    private String mParam2;

    public WorkoutDetailFragment() {
        // Required empty public constructor
    }

    /**
     * Use this factory method to create a new instance of
     * this fragment using the provided parameters.
     *
     * @param param1 Parameter 1.
     * @param param2 Parameter 2.
     * @return A new instance of fragment WorkoutDetailFragment.
     */
    // TODO: Rename and change types and number of parameters
    public static WorkoutDetailFragment newInstance(String param1, String param2) {
        WorkoutDetailFragment fragment = new WorkoutDetailFragment();
        Bundle args = new Bundle();
        args.putString(ARG_PARAM1, param1);
```

```
        args.putString(ARG_PARAM2, param2);
        fragment.setArguments(args);
        return fragment;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (getArguments() != null) {
            mParam1 = getArguments().getString(ARG_PARAM1);
            mParam2 = getArguments().getString(ARG_PARAM2);
        }
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_workout_detail, container, false);
    }

    public void setWorkout(long id) {
        this.workoutId = id;
    }
    @Override
    public void onStart() {
        super.onStart();
        View view = getView();
        if (view != null) {
            TextView title = (TextView) view.findViewById(R.id.textTitle);
            Workout workout = Workout.workouts[(int) workoutId];
            title.setText(workout.getName());
            TextView description = (TextView) view.findViewById(R.id.textDescription);
            description.setText(workout.getDescription());
        }
    }

}
```

```
package com.example.workout;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.view.LayoutInflater;
import android.view.ViewGroup;
```

```
public class WorkoutDetailFragment extends Fragment{
   private long workoutId;

   @Override
   public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
       return inflater.inflate(R.layout.fragment_workout_detail, container,
false);
   }

   public void setWorkout(long id) {
       this.workoutId = id;
   }
}
```

To get the DetailActivity to call the fragment's setWorkout() method and pass it the ID of a particular workout, the activity must get a reference to the fragment using the activity's Fragment manager. The fragment manager is used to keep track of and deal with any fragments used by the activity. There are two methods for getting a reference to the fragment manager:

- The *getSupportFragmentManager()* method gets a reference to the fragment manager that deals with fragments from the Support Library.
- The *getFragmentManager()* method gets a reference to the fragment manager that deals with fragments that use the native Android fragment class instead.

The fragment manager's findFragmentById() method is used to get a reference to the fragment as:

`getSupportFragmentManager().findFragmentById(R.id.fragment_id)`

Questions:

How findFragmentById() is different from findViewById()?

12. To make DetailActivity refer to its WorkoutDetailFragment, assign an ID `@+id/detail_frag` to an activity's fragment in the activity's layout. Update activity_detail.xml file.
13. setWorkout() from DetailActivity: Call the fragment's setWorkout() method to tell the fragment which workout we want it to display details for. For now, just **hardcode which workout to display.**

DetailActivity.java

```
package com.example.workout;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

public class DetailActivity extends AppCompatActivity {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_detail);
       WorkoutDetailFragment frag = (WorkoutDetailFragment)
           getSupportFragmentManager().findFragmentById(R.id.detail_frag);
       frag.setWorkout(1);
   }
```

```
}
```

14. Set the view's values in the WorkoutDetailFragment's onStart() method. Fragments are distinct from activities, and therefore don't have all the methods that activity does. To get a reference to a fragment's views, first get a reference to the fragment's root view using the getView() method and use that to find its child views. You should always call up to the superclass when you implement any fragment lifecycle methods.

WorkoutDetailFragment.java

```
package com.example.workout;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.view.LayoutInflater;
import android.view.ViewGroup;

public class WorkoutDetailFragment extends Fragment{
    private long workoutId;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_workout_detail, container,
false);
    }

    @Override
public void onStart() {
    super.onStart();
    View view = getView();
    if (view != null) {
        TextView title = (TextView) view.findViewById(R.id.textTitle);
        Workout workout = Workout.workouts[(int) workoutId];
        title.setText(workout.getName());
        TextView description = (TextView)
view.findViewById(R.id.textDescription);
        description.setText(workout.getDescription());
    }
}


    public void setWorkout(long id) {
        this.workoutId = id;
    }
}
```

Questions:
How fragments are different from activity and what methods which are there in activity are not there in Fragments?

Run the App

Task 2: WorkoutListFragment

We have already seen how to add a listview to an activity, so we could do something similar for the fragment. But rather than create a new fragment with a layout that contains a list view, we're going to use a different approach that involves a new type of fragment called a ListFragment. The fragment is automatically bound to a list view, so we don't need to create one. Also List fragments define their own layout programmatically, so there's no XML layout for us to create or maintain.

1. Create a fragment: Now add a new fragment called WorkoutListFragment to the project to display details of a single workout. To create the new fragment, choose **File → New → Fragment -> Fragment(Blank). Fragment(List)** can also be used but with generate code more complex.
2. Change WorkoutListFragment.java for the class to **extend ListFragment** and not Fragment.
3. Array Adapter: We want to supply the list view in WorkoutListFragment with an array of workout names, so we'll use an array adapter to bind the array to the list view as before. In Labsheet 4, we used an array adapter to display data in an activity. But this time, we want to display data in a fragment. The difference is that a fragment isn't a subclass of Context. It has no access to global information, and we can't use this to pass the current context to the array adapter. Instead, we need to get the current context in some other way. If we create the adapter in the fragment's onCreateView() method, we can use the getContext() method of the onCreateView() LayoutInflator parameter to get the context instead.

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    // Inflate the layout for this fragment
    String[] names = new String[Workout.workouts.length];
    for (int i = 0; i < names.length; i++) {
        names[i] = Workout.workouts[i].getName();
    }
    ArrayAdapter<String> adapter = new ArrayAdapter<>(
            inflater.getContext(), android.R.layout.simple_list_item_1,
            names);
    setListAdapter(adapter);
    return super.onCreateView(inflater, container, savedInstanceState);

}
```

4. Add fragment to MainActivity: Change the layout so that it displays WorkoutListFragment instead of the button. As we want MainActivity's layout to only contain a single fragment, we can get rid of the Layout tag and just add the fragment element.

   Run the app to see the layout.

Questions:
How is setListAdapter() different from ListView.setAdapter()?

Task 3: Decoupling Fragments

We are going to code for the following

- Add code to WorkoutListFragment that waits for a workout to be clicked.

- When that code runs, call some code in MainActivity.java that will start DetailActivity, passing it the ID of the workout.
- Get DetailActivity to pass the ID to WorkoutDetailFragment so that the fragment can display details of the correct workout.

We want the fragments to be reusable hence the **fragment should not know about the activity that contains it.** We have two objects that need to talk to each other—the fragment and the activity—and we want them to talk without one side knowing too much about the other for that we will use a ***Java interface*** **to decouple the fragment from the activity.**

We're going to create an interface called Listener. If MainActivity implements the interface, WorkoutListFragment will be able to tell MainActivity when one of its items has been clicked. To do this, we'll need to make changes to WorkoutListFragment and MainActivity. These are the following steps you need to follow whenever you have a fragment that needs to communicate with the activity it's attached to.

1. Listener Interface: Any activities that implement the Listener interface must include the itemClicked() method.

```
interface Listener {
    void itemClicked(long id);
}
```

2. Register the listener: We need to save a reference to the activity WorkoutListFragment gets attached to so we will use the fragment's onAttach() Method [Fragment Lifecycle]

```
static interface Listener {
    void itemClicked(long id);
};
private Listener listener;

@Override
public void onAttach(Context context) {
    super.onAttach(context);
    this.listener = (Listener)context;
}

@Override
public void onListItemClick(ListView listView, View itemView, int position,
long id) {
    if (listener != null) {
        listener.itemClicked(id);
    }
}
```

3. Implement the interface in MainActivity:

public class MainActivity extends AppCompatActivity implements WorkoutListFragment.Listener {}

Implement the itemClicked method to Start DetailActivity. Use `intent.putExtra(DetailActivity.EXTRA_WORKOUT_ID, (int)id);`

to pass the workout id.

4. Pass the ID from Activity to Fragment: Now that the DetailsActivity has the information about the workout id, pass it to the fragment using frag.setWorkout(workoutId);

Questions:
Can you list down all the events that happen since the app is launched?

Extra exercise: Fragment lifecycle
Fragments have their own life cycle very similar to an Activity but with extra events that are particular to the Fragment's view hierarchy, state, and attachment to its activity.

Create a demo app to demonstrate Fragment LIfecycle in Android. For example, create an activity with one fragment and override its methods to display a message using logs.

```
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
printLog("onCreate Called");
}
```