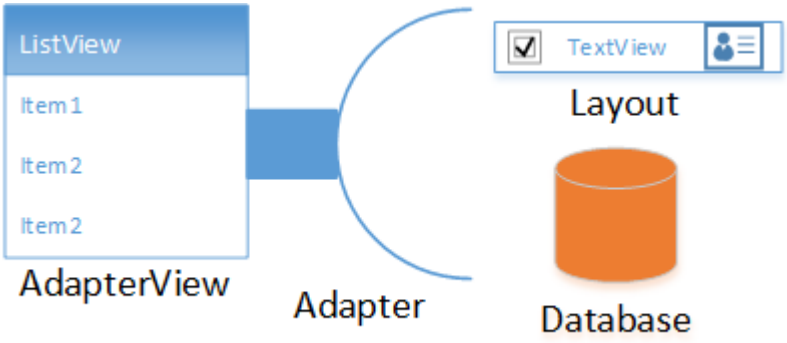
MOBILE APPLICATION DEVELOPMENT (MAD) – LAB 7

Objectives:

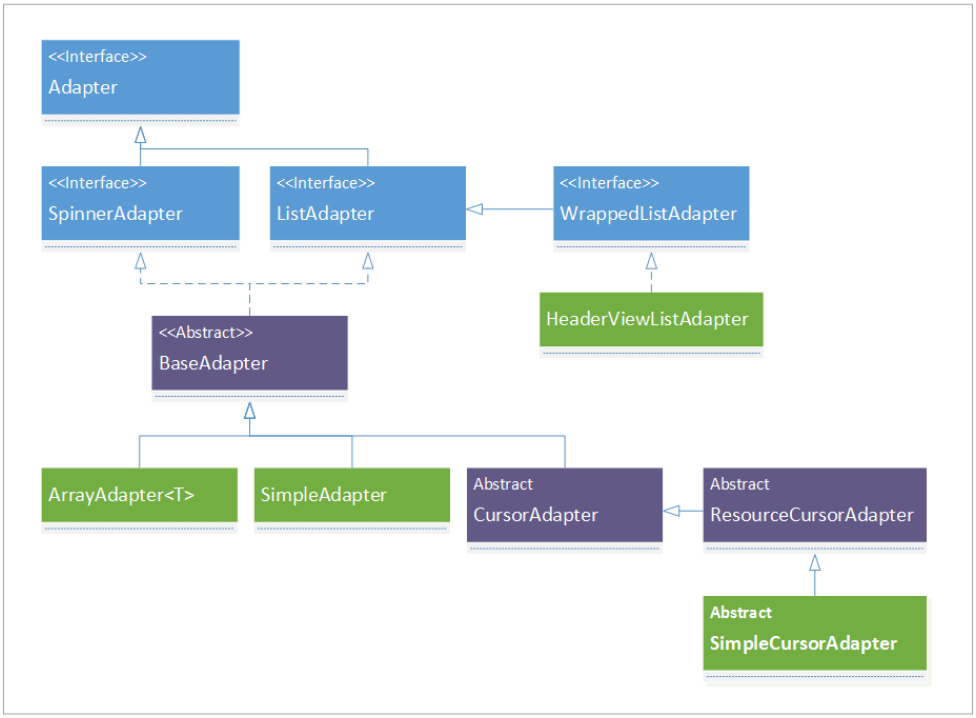
- Understanding the usage of Adapters
- Understanding Fragments
- Practice Activities

Objective 1: Introduction to Adapters.

- A bridge between an AdapterView and the underlying data for that view.
- An AdapterView is a group of widgets (aka view) components in Android that include the ListView, Spinner, and GridView.
- AdapterView also provides the layout of the underlying data for the view



Types of Adapters

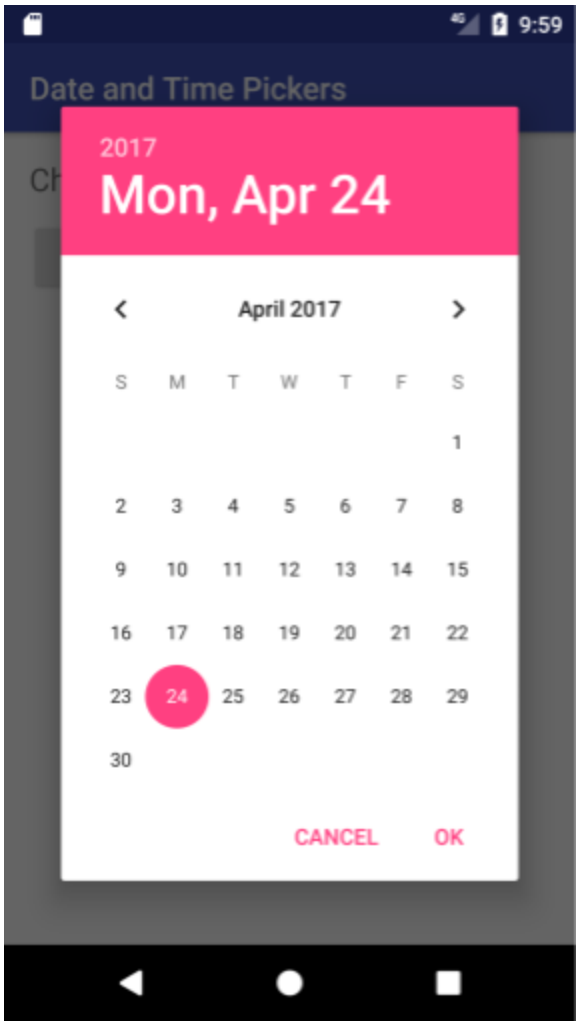


Adapter View Methods

Fragment with View Pager (WhatsApp like fragments):
https://github.com/codepath/android_guides/wiki/ViewPager-with-FragmentPagerAdapter

- **getCount( ):** indicates to Android how many items (or rows) are in the data set that will be presented in the AdapterView.
- **getItem(int pos):** get the data item associated with the item (or row) from the AdapterView passed as a parameter to the method. This method will be used by Android to fetch the appropriate data to build the item/row in the AdapterView.
- **getItemId(int pos):** This method returns the data set's id for a item/row position of the AdapterView. Typically, the data set id matches the AdapterView rows so this method just returns the same value.
- **getView(int position, View convertView, ViewGroup parent):** This method creates the View (which may be a single View component like a TextView or a complex set of widgets in a layout) that displays the data for the specified (by position) item/row in the AdapterView

OBJECTIVE 2: Introduction to Fragments



A Fragment is a self-contained component with its own user interface (UI) and lifecycle that can be reused in different parts of an app's UI. (A Fragment can also be used without a UI, in order to retain values across configuration changes, but this lesson does not cover that usage.)

A Fragment can be a *static* part of the UI of an Activity, which means that the Fragment remains on the screen during the entire lifecycle of the Activity. However, the UI of an Activity may be more effective if it adds or removes the Fragment *dynamically* while the Activity is running.

One example of a dynamic Fragment is the DatePicker object, which is an instance of DialogFragment, a subclass of Fragment. The date picker displays a dialog window floating on top of its Activity window when a user taps a button or an action occurs. The user can click **OK** or **Cancel** to close the Fragment.

Importance of Fragments

There are many use cases for fragments but the most common use cases include:

- **Reusing View and Logic Components** - Fragments enable re-use of parts of your screen including views and event logic over and over in different ways across many disparate activities. For example, using the same list across different data sources within an app.
- **Tablet Support** - Often within apps, the tablet version of an activity has a substantially different layout from the phone version which is different from the TV version. Fragments enable device-specific activities to reuse shared elements while also having differences.

Fragment with View Pager (WhatsApp like fragments):
https://github.com/codepath/android_guides/wiki/ViewPager-with-FragmentPagerAdapter

- **Screen Orientation** - Often within apps, the portrait version of an activity has a substantially different layout from the landscape version. Fragments enable both orientations to reuse shared elements while also having differences.

Embedding a Fragment in an Activity

There are two ways to add a fragment to an activity: dynamically using **Java** and statically using **XML**.

Before embedding a "support" fragment in an Activity make sure the Activity is changed to extend from FragmentActivity or AppCompatActivity which adds support for the fragment manager to all Android versions. Any activity using fragments should make sure to extend from FragmentActivity or AppCompatActivity:

➔ Statically

To add the fragment statically, simply embed the fragment in the activity's xml layout file:

```
<?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="vertical" >

    <fragment
        android:name="com.example.android.FooFragment"
        android:id="@+id/fooFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

Note:

- You will likely need to change the path for FooFragment based on your project setup.
- You cannot replace a fragment defined statically in the layout file via a FragmentTransaction. You can only replace fragments that you added dynamically

Fragment with View Pager (WhatsApp like fragments):
https://github.com/codepath/android_guides/wiki/ViewPager-with-FragmentPagerAdapter

➔ Dynamically

The second way is by adding the fragment **dynamically** in Java using the **FragmentManager**.
The **FragmentManager** class and the FragmentTransaction class allow you to add, remove and replace fragments in the layout of your activity at runtime.

In this case, you want to add a "placeholder" container (usually a **FrameLayout**) to your activity where the fragment is inserted at runtime:

```
<?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="vertical" >

  <FrameLayout
       android:id="@+id/your_placeholder"
       android:layout_width="match_parent"
       android:layout_height="match_parent">
  </FrameLayout>

</LinearLayout>
```

and then you can use the FragmentManager to create a FragmentTransaction which allows us to add fragments to the FrameLayout at runtime:

```
// Begin the transaction
FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
// Replace the contents of the container with the new fragment
ft.replace(R.id.your_placeholder, new FooFragment());
// or ft.add(R.id.your_placeholder, new FooFragment());
// Complete the changes added above
ft.commit();
```

Communicating with Fragments

Fragments should generally only communicate with their direct parent activity. Fragments communicate through their parent activity allowing the activity to manage the inputs and outputs of data from that fragment coordinating with other fragments or activities. Think of the Activity as the controller managing all interaction with each of the fragments contained within.

A few exceptions to this are dialog fragments presented from within another fragment or nested child fragments. Both of these cases are situations where a fragment has nested child fragments and that are therefore allowed to communicate upward to their parent (which is a fragment).

The important thing to keep in mind is that **fragments should not directly communicate with each other** and should generally only **communicate with their parent activity**. Fragments should be modular,

Fragment with View Pager (WhatsApp like fragments):
https://github.com/codepath/android_guides/wiki/ViewPager-with-FragmentPagerAdapter

standalone and reusable components. The fragments allow their parent activity to respond to intents and callbacks in most cases.

There are three ways a fragment and an activity can communicate:

1. **Bundle** - Activity can construct a fragment and set arguments
2. **Methods** - Activity can call methods on a fragment instance
3. **Listener** - Fragment can fire listener events on an activity via an interface

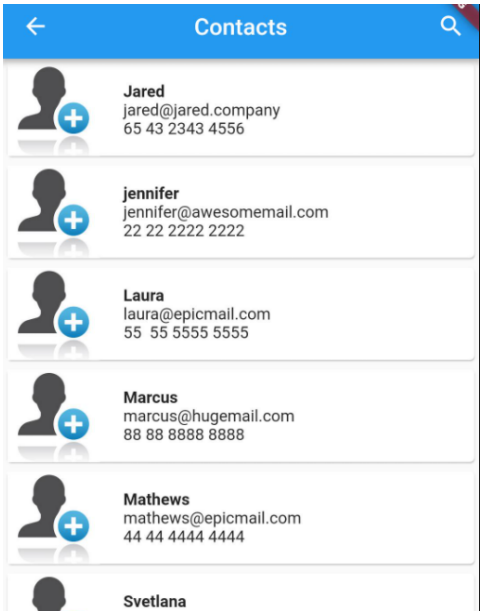
In other words, communication should generally follow these principles:

- Activities can initialize fragments with data during construction
- Activities can pass data to fragments using methods on the fragment instance
- Fragments can communicate up to their parent activity using an interface and listeners
- Fragments should pass data to other fragments only routed through their parent activity
- Fragments can pass data to and from dialog fragments as outlined here
- Fragments can contain nested child fragments as outlined here

Fragment with View Pager (WhatsApp like fragments):
https://github.com/codepath/android_guides/wiki/ViewPager-with-FragmentPagerAdapter

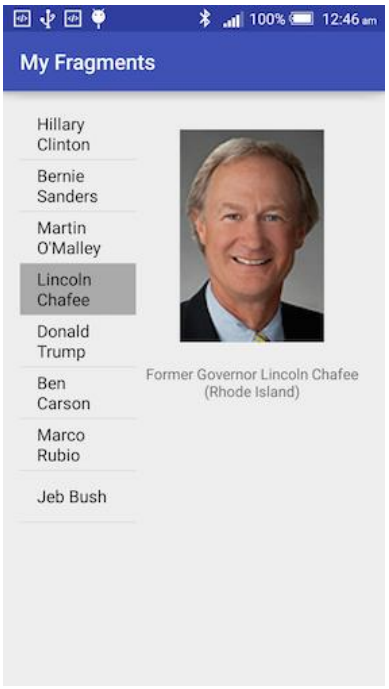
OBJECTIVE 3: ACTIVITIES.

Activity 1: Modify the Task 2 – Lab 6, show the contacts in a listview. Name, Phone Number and email along with a sample image on left side of the contact.



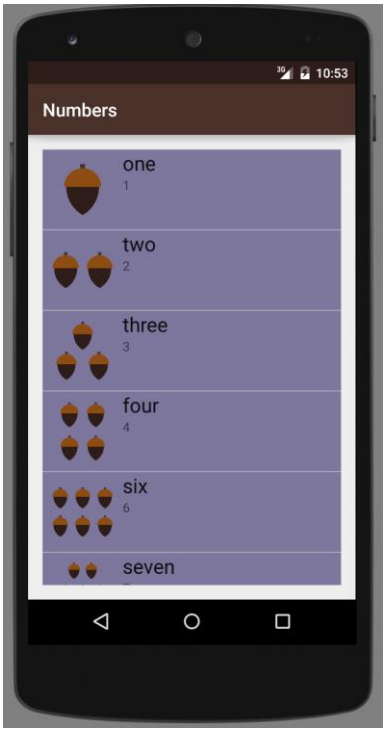
Fragment with View Pager (WhatsApp like fragments):
https://github.com/codepath/android_guides/wiki/ViewPager-with-FragmentPagerAdapter

Activity 2: Create a “Presidential Election Candidates” App, which will list all the candidates being available for election, clicking on its image will load the detail of the candidate in another fragment.



Fragment with View Pager (WhatsApp like fragments):
https://github.com/codepath/android_guides/wiki/ViewPager-with-FragmentPagerAdapter

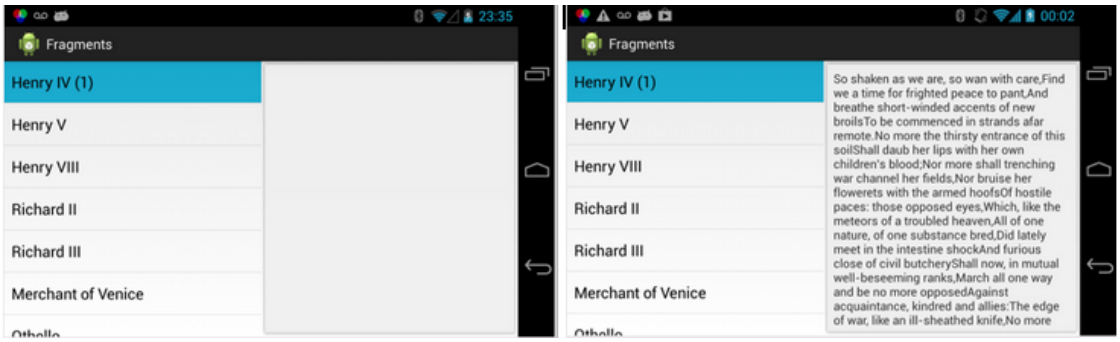
Activity 3: The Moquelumnan languages are a group of endangered languages spoken in Central California in the Sierra Nevada. In this exercise, you will build an app to help Moquelumnan language students practice their number skills. The screen, as shown below, will provide a list of number of items from 1 to 10. For each number provide an image, the written English name, the numeric value, and an audio sound that will play when the user taps directly on the list item. Use a ListView powered by a custom ArrayAdapter



Fragment with View Pager (WhatsApp like fragments):
https://github.com/codepath/android_guides/wiki/ViewPager-with-FragmentPagerAdapter

MOBILE APPLICATION DEVELOPMENT (MAD) – LAB 7

Activity 4: Modify Activity 1 – Lab 6, let the user able to cycle through the stories loaded in a list from database. And able to read any story when user clicks on it.



Fragment with View Pager (WhatsApp like fragments):
https://github.com/codepath/android_guides/wiki/ViewPager-with-FragmentPagerAdapter