MOBILE APPLICATION DEVELOPMENT (MAD) – LAB 6

Objectives:

- To understand the file handling
- Understanding the usage of Adapters
- Understanding List View
- Understanding Fragments
- Practice Activities

OBJECTIVE 1: Understanding File Storage on Android

- Android uses a file system that's similar to disk-based file systems on other platforms.
- A `File` object works well for reading or writing large amounts of data in start-to-finish order without skipping around.

➢ **Choose Where to store data**

Internal storage:

- It's always available.
- Files saved here are accessible by only your app.
- When the user uninstalls your app, the system removes all your app's files from internal storage.

Internal storage is best when you want to be sure that neither the user nor other apps can access your files.

External storage:

- It's not always available, because the user can mount the external storage as USB storage and in some cases remove it from the device.
- It's world-readable, so files saved here may be read outside of your control.
- When the user uninstalls your app, the system removes your app's files from here only if you save them in the directory from `getExternalFilesDir()`.

➢ **Saving data on Internal Storage**

You can acquire the appropriate directory as a File by calling one of two methods:

- **getFilesDir()**
  Returns a File representing an internal directory for your app.
- **getCacheDir()**
  Returns a File representing an internal directory for your app's temporary cache files. Be sure to delete each file once it is no longer needed and implement a reasonable size limit for the amount of memory you use at any given time, such as 1MB.

```
String filename = "myfile";
String fileContents = "Hello world!";
FileOutputStream outputStream;

try {
    outputStream = openFileOutput(filename, Context.MODE_PRIVATE);
    outputStream.write(fileContents.getBytes());
    outputStream.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

➢ **Opening a file from Internal Storage**

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

```
File directory = context.getFilesDir();
File file = new File(directory, filename);
```

➢ **Saving data on External Storage**

After you request storage permissions and verify that storage is available, you can save two different types of files:

- **Public files**: Files that should be freely available to other apps and to the user. When the user uninstalls your app, these files should remain available to the user. For example, photos captured by your app or other downloaded files should be saved as public files.
- **Private files**: Files that rightfully belong to your app and will be deleted when the user uninstalls your app. Although these files are technically accessible by the user and other apps because they are on the external storage, they don't provide value to the user outside of your app

➢ **Request permission before accessing external storage**

```
<manifest ...>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    ...
</manifest>
```

```
<manifest ...>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    ...
</manifest>
```

➢ **Methods to get access to the storage**

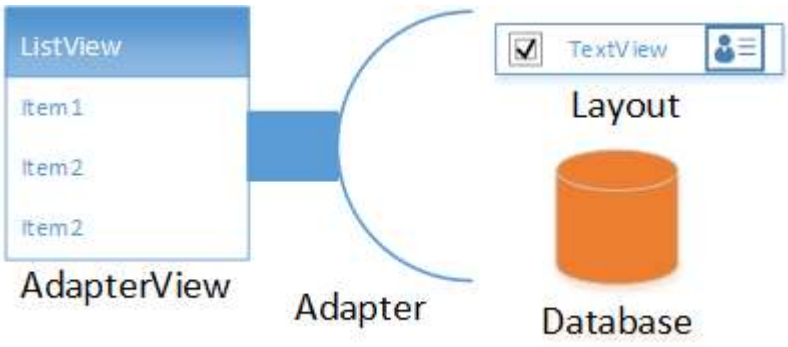
- **getExternalFilesDir()**
  getExternalFilesDir() creates a directory that is deleted when the user uninstalls your app.

- **getExternalStoragePublicDirectory()**
  You use **getExternalStoragePublicDirectory()** for files that are shared and will not be deleted when app is uninstalled.

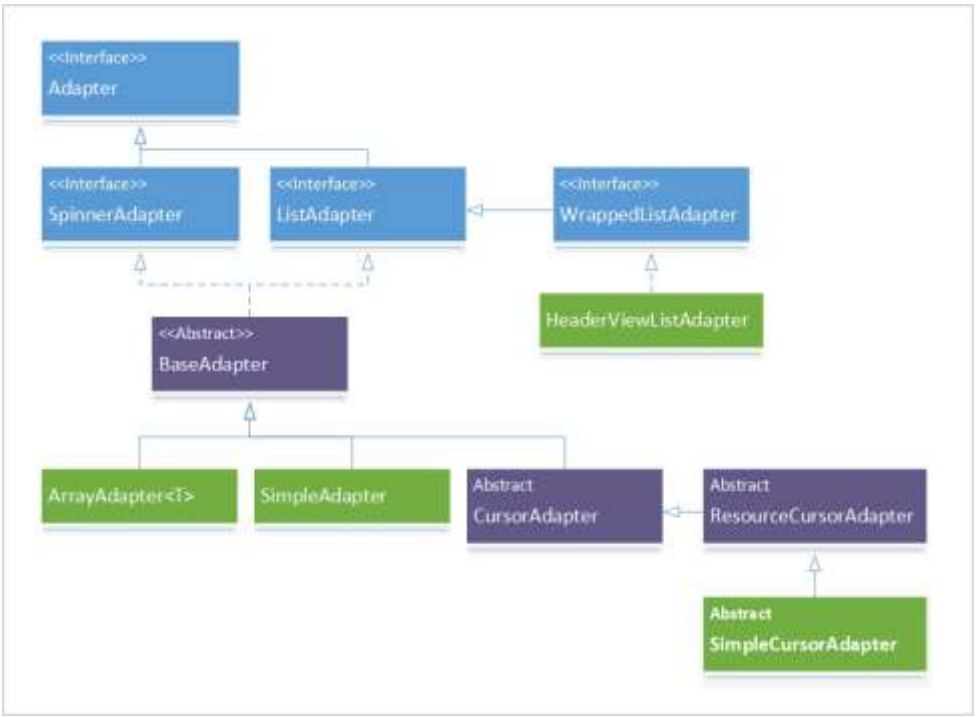
Fragment with View Pager (WhatsApp like fragments):
https://github.com/codepath/android_guides/wiki/ViewPager-with-FragmentPagerAdapter

Objective 2: 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

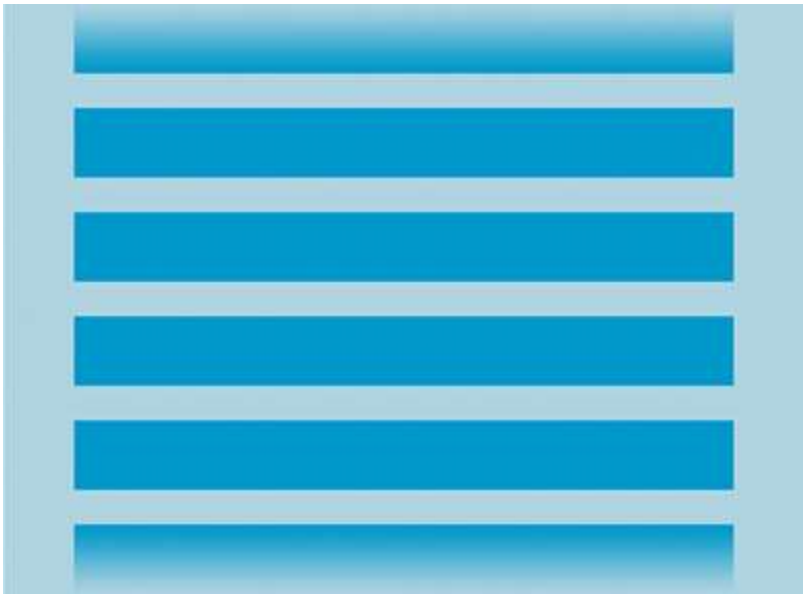
- **getCount( ):** indicates to Android how many items (or rows) are in the data set that will be presented in the AdapterView.
- **getItem(int pos):** get the data item associated with the item (or row) from the AdapterView passed as a parameter to the method. This method will be used by Android to fetch the appropriate data to build the item/row in the AdapterView.

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

- **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 3: Introduction to ListView

- **ListView** is a view group that displays a list of scrollable items. The list items are automatically inserted to the list using an Adapter that pulls content from a source such as an array or database query and converts each item result into a view that's placed into the list.



- To display a more custom view for each item in your dataset, implement a **ListAdapter**. For example, extend **BaseAdapter** and create and configure the view for each data item in **getView**(...):

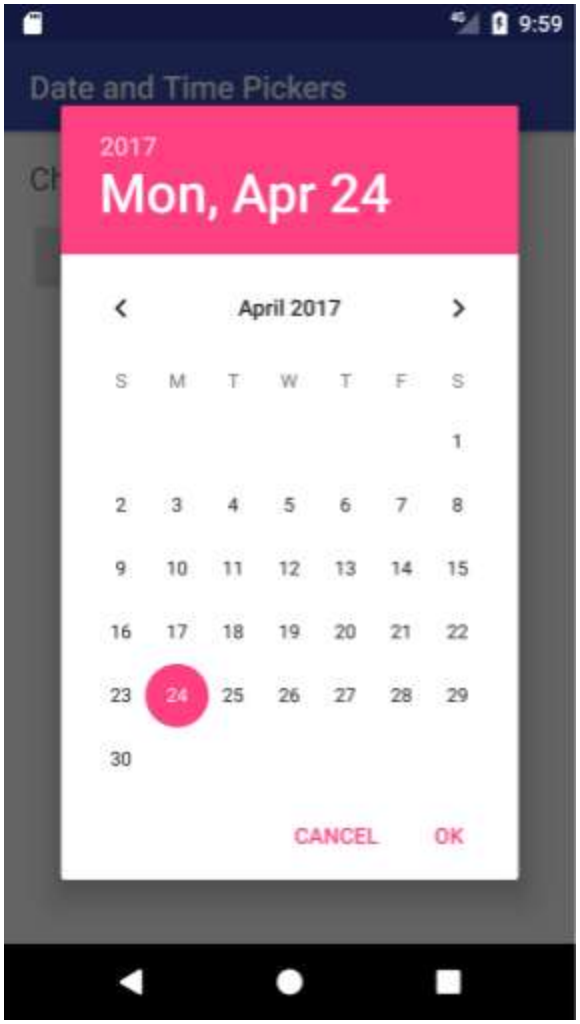
Fragment with View Pager (WhatsApp like fragments):
https://github.com/codepath/android_guides/wiki/ViewPager-with-FragmentPagerAdapter

```
private class MyAdapter extends BaseAdapter {

    // override other abstract methods here

    @Override
    public View getView(int position, View convertView, ViewGroup container) {
        if (convertView == null) {
            convertView = getLayoutInflater().inflate(R.layout.list_item, container, false);
        }

        ((TextView) convertView.findViewById(android.R.id.text1))
                .setText(getItem(position));
        return convertView;
    }
}
```



OBJECTIVE 4: 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.

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

- **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.
- **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: Create a game MadLibs.

"Mad Libs" are short stories that have blanks called placeholders to be filled in. In the non-computerized version of this game, one person asks a second person to fill in each of the placeholders without the second person knowing the overall story. Once all placeholders are filled in, the second person is shown the resulting silly story.

Write an Android app that reads in a Mad Lib from a text file in a specific format. The text file represents placeholders as tokens that start and end with < > brackets, like <adjective> or <proper-noun>. Your app reads the file, looks for any such placeholders, and prompts the user to replace them with specific words. Once the user has typed in replacements for all placeholders, the completed story is shown on the screen. The screenshots below indicate a possible flow of the UI for such an app. Our flow has three activities: An initial "welcome" screen explaining the app, then a screen that repeatedly prompts the user to fill in placeholders, then a third activity to display the completed story. Of course you don't need to exactly match our sample's UI, but it may give you ideas.

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

Activity 2: Add contacts File : Steps to follow

Steps	Description
1	You will use Android studio to create an Android application under a package com.example.sairamkrishna.myapplication.
2	Modify src/MainActivity.java file to get references of all the XML components and populate the contacts on listView.
3	Create new src/DBHelper.java that will manage the database work
4	Create a new Activity as DisplayContact.java that will display the contact on the screen
5	Modify the res/layout/activity_main to add respective XML components
6	Modify the res/layout/activity_display_contact.xml to add respective XML components
7	Modify the res/values/string.xml to add necessary string components
8	Modify the res/menu/display_contact.xml to add necessary menu components
9	Create a new menu as res/menu/mainmenu.xml to add the insert contact option
10	Run the application and choose a running android device and install the application on it and verify the results.

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

Activity 3: Modify TODO – List app of Week 4 / Activity 2.

- Add all the to-do list items in the file instead of storing them in memory.

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