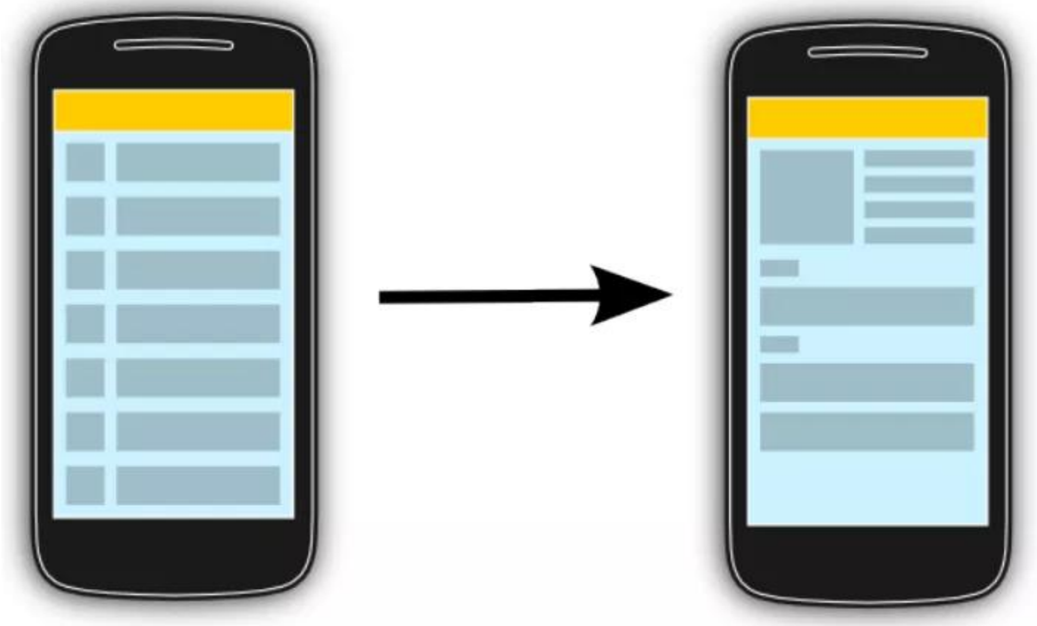
Introduction to Mobile Computing (week 5)

Dieudonne U

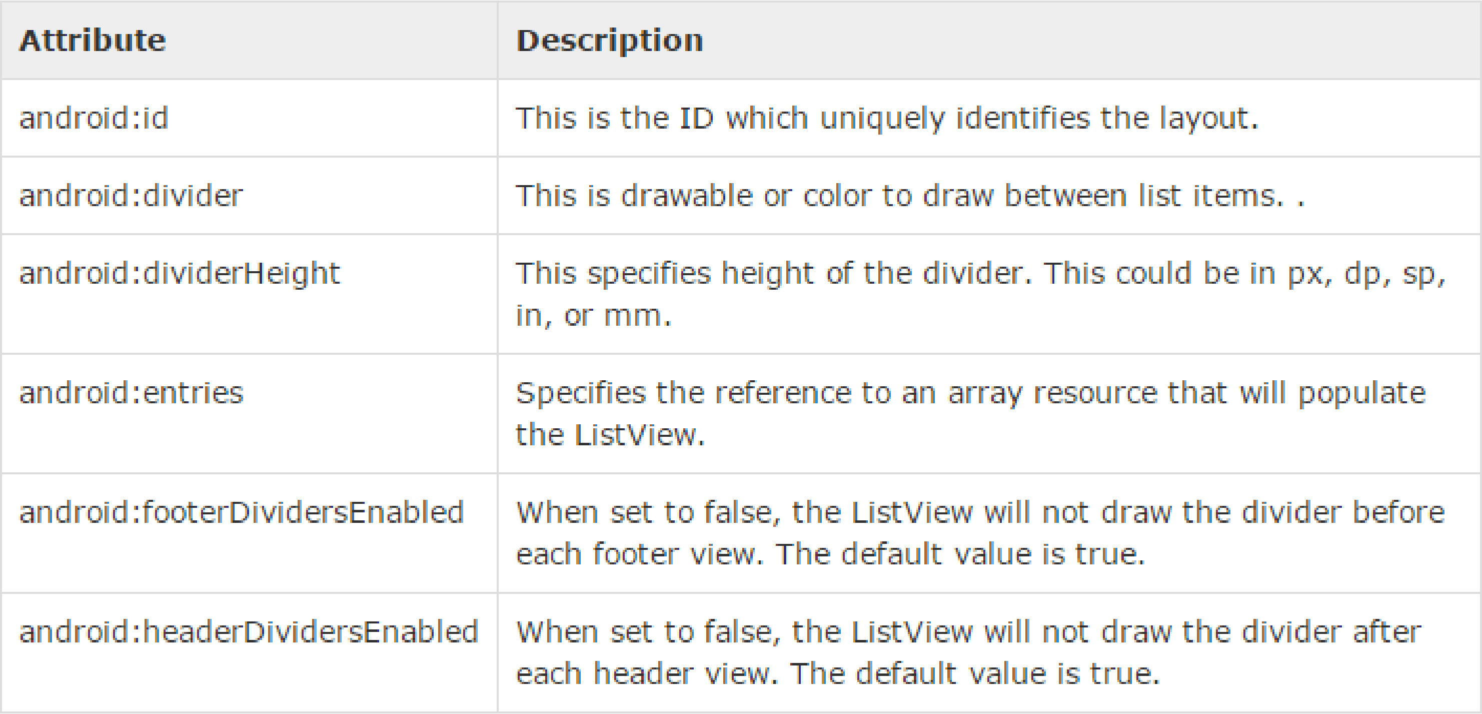
Lists in Android

- The display of elements in a list is a very common pattern in mobile applications.
- The user sees a list of items and can scroll through them. Such an activity is depicted in the following picture



ListView widget

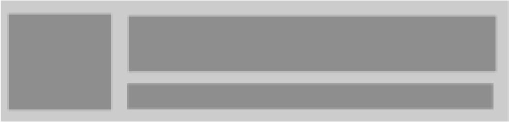
- ListView is a view group that displays a list of scrollable items or in other words is a view which groups several items and display them in vertical scrollable list



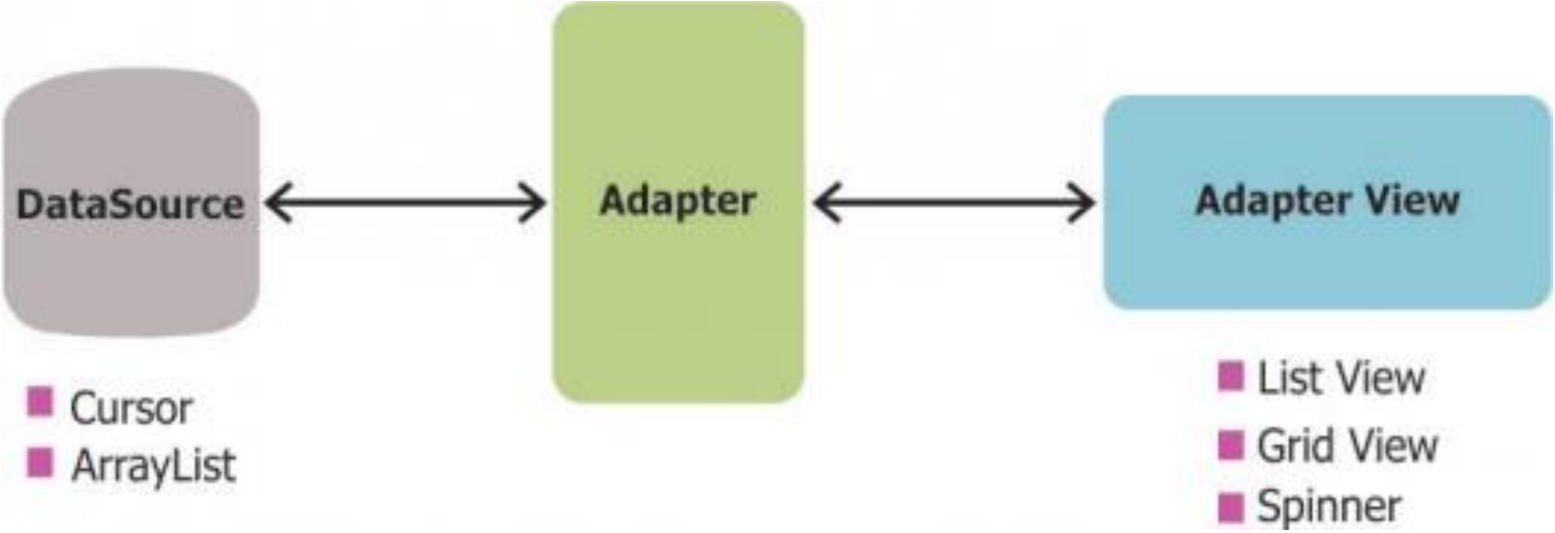
Attribute	Description
android:id	This is the ID which uniquely identifies the layout.
android:divider	This is drawable or color to draw between list items. .
android:dividerHeight	This specifies height of the divider. This could be in px, dp, sp, in, or mm.
android:entries	Specifies the reference to an array resource that will populate the ListView.
android:footerDividersEnabled	When set to false, the ListView will not draw the divider before each footer view. The default value is true.
android:headerDividersEnabled	When set to false, the ListView will not draw the divider after each header view. The default value is true.

Adapters

- 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
- An *adapter* manages the data model and adapts it to the individual entries in the widget. An adapter extends the BaseAdapter class.
- Every line in the widget displaying the data consis which can be as complex as you want.



- A typical line in a list has an image on the left side and two text lines in the middle as depicted in the following graphic.



ArrayAdapter

- By default, ArrayAdapter creates a view for each array item by calling toString() on each item and placing the contents in a **TextView**.
- Consider you have an array of strings you want to display in a ListView, initialize a new **ArrayAdapter** using a constructor to specify the layout for each string and the string array –

```
ArrayAdapter adapter =
            new ArrayAdapter<String>(this,R.layout.ListView,R.Row, StringArray);
```

ListView Using Custom Adapter

- So far we have created a simple list view using ArrayAdapter. Now it is time to create custom a list by extending BaseAdapter.
- First step towards building custom list is to identify the data model for each row. In our example we will display list of Student objects.
- Secondly, let us declare list view in activity layout.
- Now declare the layout for each row item.
- Create your custom adapter class by extending BaseAdapter class.
- Finally, let us instantiate your custom adapter and set to ListView by calling setAdapter() method.

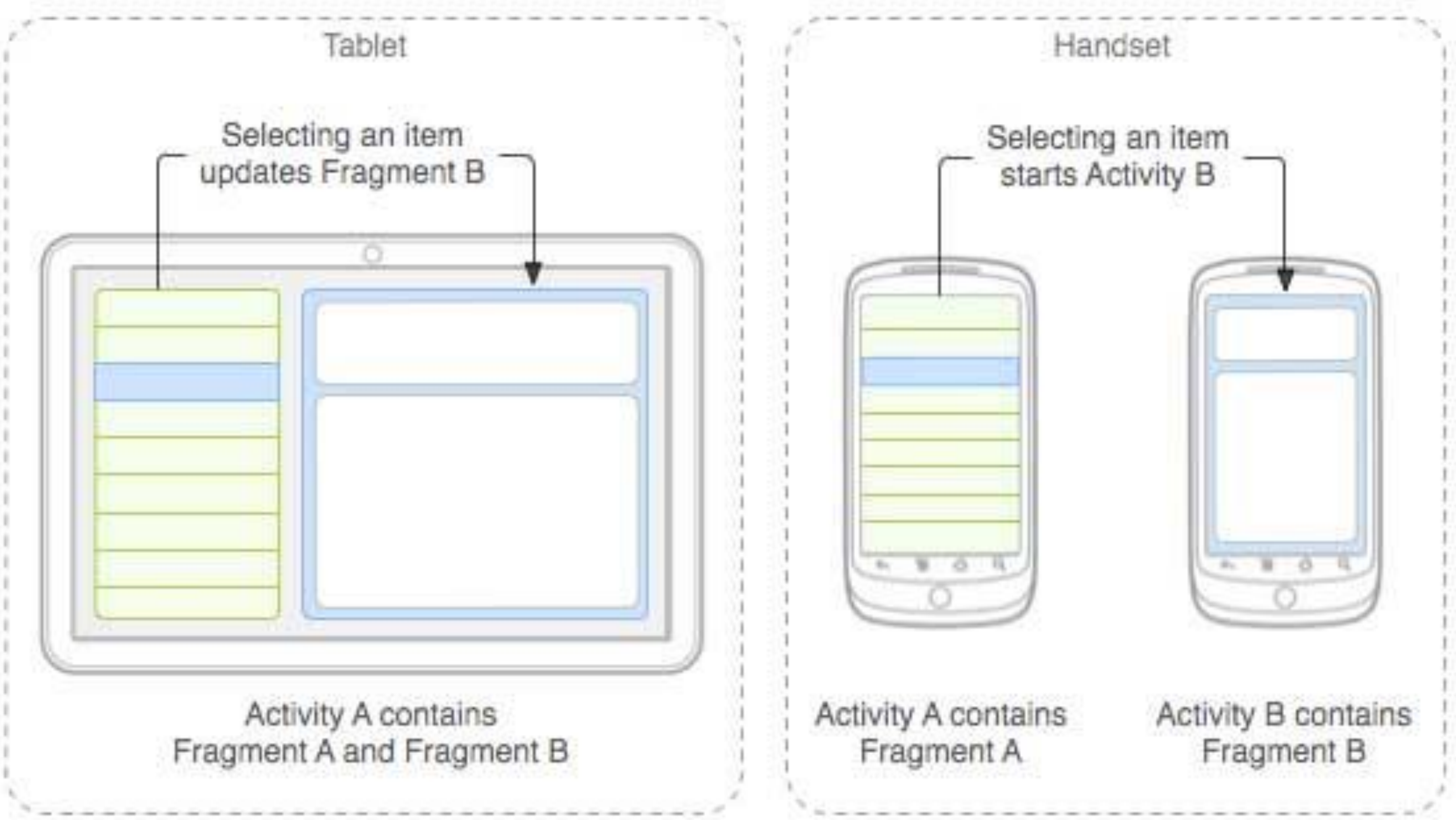
Fragments

- An Android fragment is a GUI component which can "live" inside an Activity.
- An Android fragment is not by itself a subclass of View which most other GUI components are. Instead, a fragment has a view inside it.
- It is this view which is eventually displayed inside the activity in which the fragment lives.
- Because an Android fragment is not a view, adding it to an activity looks somewhat different than adding a view (e.g. TextView).

Important points about fragment

- A fragment has its own layout and its own behaviour with its own life cycle callbacks.
- You can add or remove fragments in an activity while the activity is running.
- You can combine multiple fragments in a single activity to build a multi-plane UI.
- A fragment can be used in multiple activities.
- Fragment life cycle is closely related to the life cycle of its host activity which means when the activity is paused, all the fragments available in the activity will also be stopped.
- A fragment can implement a behaviour that has no user interface component.
- Fragments were added to the Android API in Honeycomb version of Android which API version 11.

Create fragments



Tablet

Selecting an item updates Fragment B

Activity A contains Fragment A and Fragment B

Handset

Selecting an item starts Activity B

Activity A contains Fragment A

Activity B contains Fragment B

Create a Fragment

- You create fragments by extending **Fragment** class and You can insert a fragment into your activity layout by declaring the fragment in the activity's layout file, as a **<fragment>**element.

- To create a fragment you must do two things:
- Create a Fragment class.
- Create a Fragment layout XML file.

Creating a Fragment

```
public static class ExampleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.example_fragment, container, false);
    }
}
```

onCreateView()

- This Fragment subclass isn't doing anything yet. The MyFragment class needs to override the method onCreateView() inherited from Fragment in order to create the fragment's

```
public View onCreateView(LayoutInflater inflater, ViewGroup parentViewGroup,
Bundle savedInstanceState) {
View view = inflater.inflate(R.layout.fragment_my, parentViewGroup, false);
```

- The **LayoutInflater** is an component which can create View instance based on layout XML files. As you can see, the example actually does that by calling **layout.inflate**() .

Add a Fragment to an Activity

Fragments must be added (or replaced or removed) inside a FragmentTransaction. You obtain a FragmentTransaction via the FragmentManager.

```
 getFragmentManager()
   .beginTransaction()
   .add(R.id.fragmentParentViewGroup, new
   MyFragment())
.commit();
```

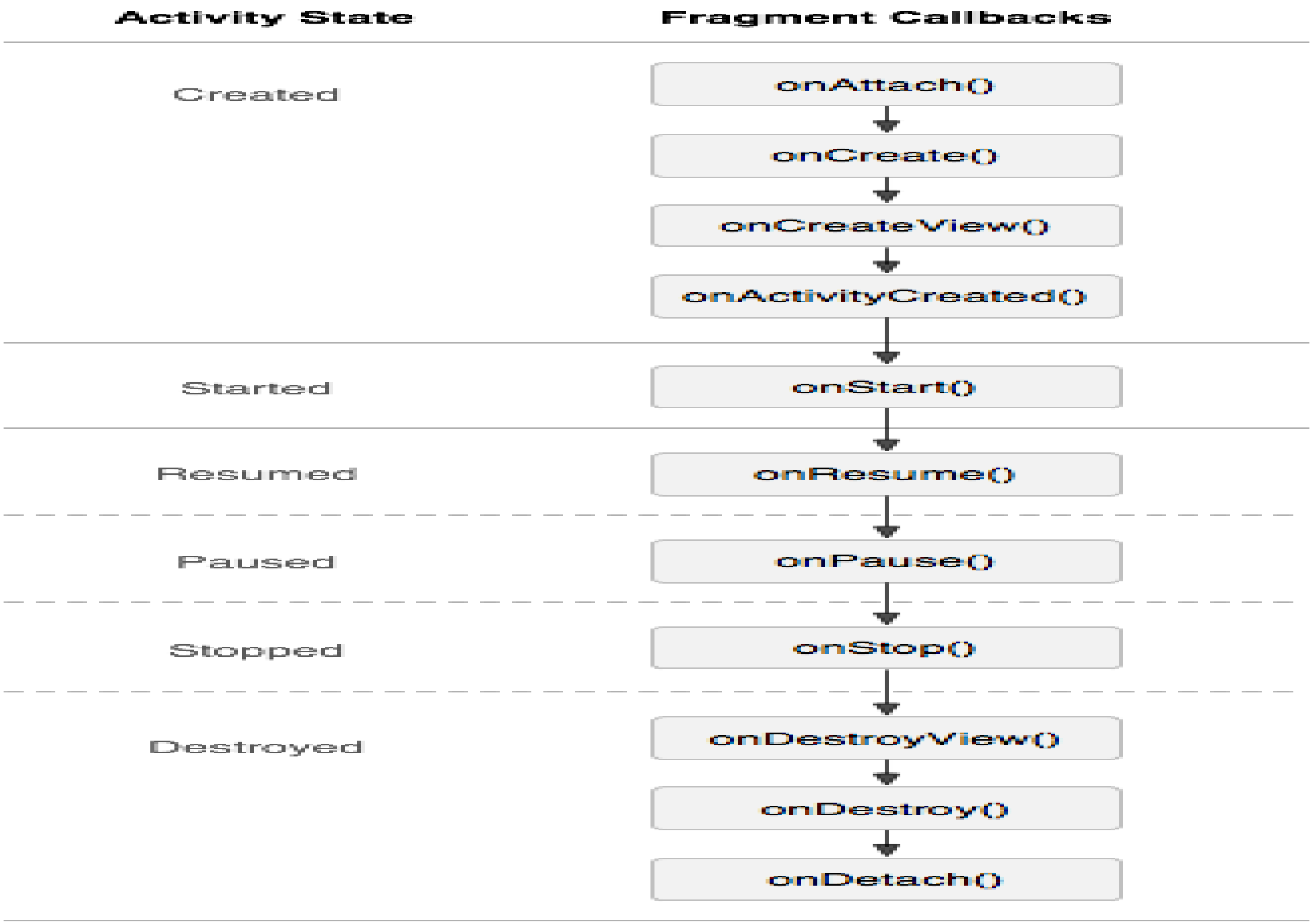
```
FragmentManager fragmentManager = getSupportFragmentManager();
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
```

You can then add a fragment using the add() method, specifying the fragment to add and the view in which to insert it. For example:

```
ExampleFragment fragment = new ExampleFragment();
fragmentTransaction.add(R.id.fragment_container, fragment);
fragmentTransaction.commit();
```

The first argument passed to add() is the ViewGroup in which the fragment should be placed, specified by resource ID, and the second parameter is the fragment to add.
Once you've made your changes with FragmentTransaction, you must call commit() for the changes to take effect.

Fragment Life cycle



Activity State	Fragment Callbacks
Created	onAttach()
	onCreate()
	onCreateView()
	onActivityCreated()
Started	onStart()
Resumed	onResume()
Paused	onPause()
Stopped	onStop()
Destroyed	onDestroyView()
	onDestroy()
	onDetach()

Managing Fragments

To manage the fragments in your activity, you need to use FragmentManager.
To get it, call getSupportFragmentManager() from your activity.

Some things that you can do with FragmentManager include:

- Get fragments that exist in the activity, with findFragmentById() (for fragments that provide a UI in the activity layout) or findFragmentByTag() (for fragments that do or don't provide a UI).

- Pop fragments off the back stack, with popBackStack() (simulating a *Back* command by the user).

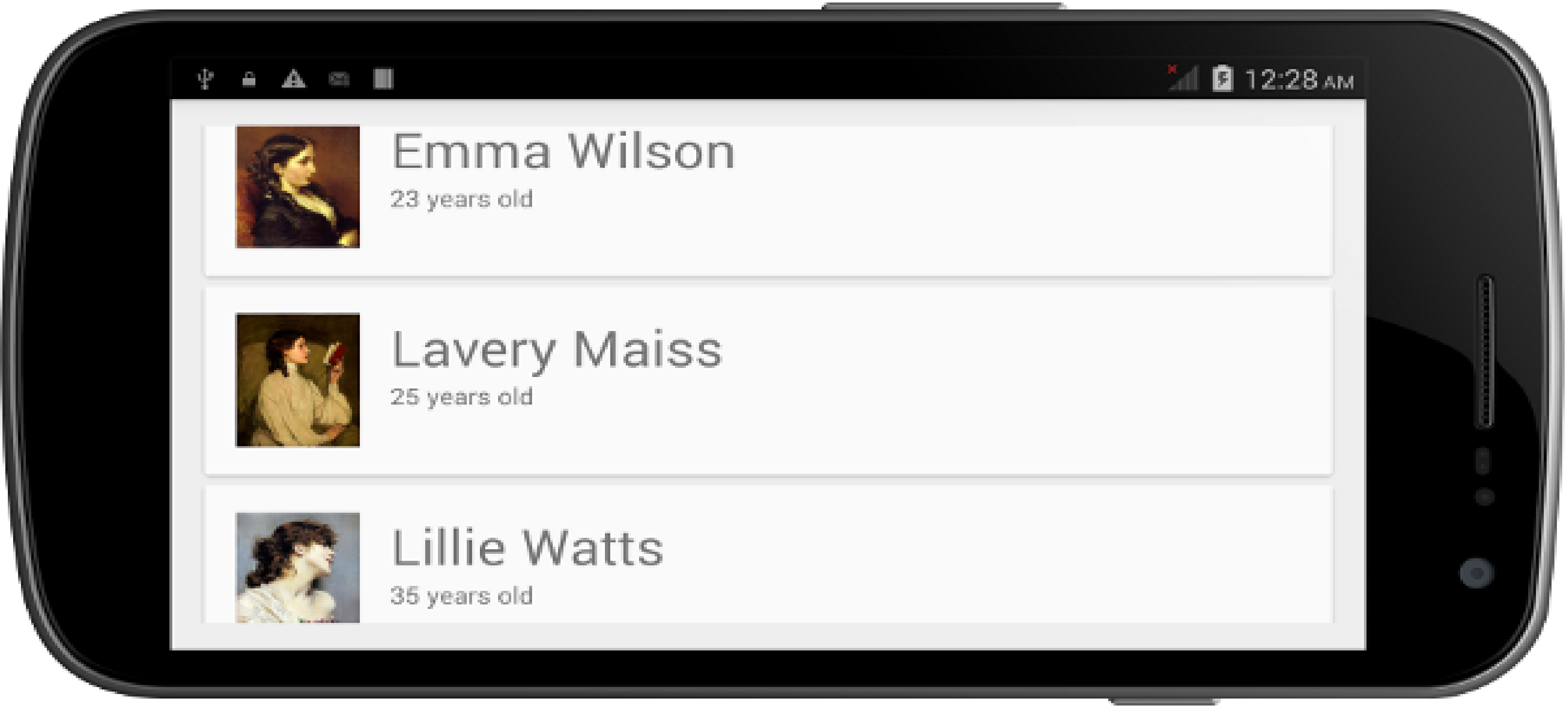
- Register a listener for changes to the back stack, with addOnBackStackChangedListener().

FragmentManager can also be used to open a FragmentTransaction, which allows you to perform transactions, such as add and remove fragments.

RecyclerView widget

- **What is RecyclerView?**

- RecyclerView is more advanced and flexible and efficient version of ListView.
- RecyclerView ViewGroup is a container for larger data set of views that can be recycled and scrolled very efficiently.
- RecyclerView can be used for larger datasets to be rendered on the UI like a list.
- RecyclerView provides maximum flexibility to design different kind of views
- Android RecyclerView falls under the material design of android



Emma Wilson
23 years old

Lavery Maiss
25 years old

Lillie Watts
35 years old

RecyclerView with different layouts

- **LinearLayoutManager** – This LayoutManager can be used to display linear lists, they could be vertical or horizontal.
- **GridLayoutManager** – Earlier in android GridView was the only widget to display grids, but now with RecyclerView, GridLayoutManager can be used to display grids.

Android RecyclerView ViewHolder

- The concept of RecyclerView.ViewHolder Simply said: when a view goes out of visible area it is kept for recycling.

- RecyclerView a ViewHolder class is included in the adapter by default and it is compulsion for everyone to implement this class.

Steps to use a RecyclerView:

- Creating a project with an activity which will display the RecylerView
  - E.g. MainActivity
  - Add recycleView
  - Create layout_row
- Add an abject model that will be used to set and get data
- Create an adapter class

onCreateViewHolder and **onBindViewHolder.**

- You have to override two method **onCreateViewHolder** and **onBindViewHolder.**
  - **onCreateViewHolder** is used to create a new RecyclerView.ViewHolder and initializes some private fields to be used by RecyclerView.
  - **onBindViewHolder(ViewHolder**, int) updates the RecyclerView.ViewHolder contents with the item at the given position and also sets up some private fields to be used by RecyclerView.
- onCreateViewHolder only creates a new view holder when there are no existing view holders which the RecyclerView can reuse.
- So, for instance, if your RecyclerView can display 5 items at a time, it will create 5-6 ViewHolders, and then automatically reuse them, each time calling onBindViewHolder.

TabLayout

- TabLayout provides a horizontal layout to display tabs.
- Population of the tabs to display is done through TabLayout.Tab instances. You create tabs via newTab(). From there you can change the tab's label or icon via setText(int) andsetIcon(int) respectively. To display the tab, you need to add it to the layout via one of the addTab(Tab) methods. For example:

Android Storage

- Android provides several options to save persistent application data.
- The solution you choose depends on your specific needs, such as whether the data should be private to your application or accessible to other applications (and the user) and how much space your data requires.
- Data storage options are the following:
  - Shared Preferences
    - Store private primitive data in key-value pairs.
  - Internal Storage
    - Store private data on the device memory.
  - External Storage
    - Store public data on the shared external storage.
  - SQLite Databases
    - Store structured data in a private database.
  - Network Connection
    - Store data on the web with your own network server.
  - Android provides a way for you to expose even your private data to other applications — with a content provider
  - Content provider. A content provider is an optional component that exposes read/write access to your application data, subject to whatever restrictions you want to impose

Shared Preferences.

- Session help you when want to store user data outside your application, so that when the next time user use your application, you can easily get back his details and perform accordingly.
- This can be done in many ways. But the most way of doing this is through **Shared Preferences**.

Using shared preferences

- You have to call a method getSharedPreferences() that returns a SharedPreference instance pointing to the file that contains the values of preferences.

```
SharedPreferences sharedpreferences =
    getSharedPreferences(MyPREFERENCES,
    Context.MODE_PRIVATE);
```

- You can save something in the sharedpreferences by using **SharedPreferences.Editor** class.
- You will call the edit method of SharedPreference instance and will receive it in an editor object.

```
Editor editor = sharedpreferences.edit();
editor.putString("key", "value");
editor.commit();
```

Using the Internal Storage

- You can save files directly on the device's internal storage. By default, files saved to the internal storage are private to your application and other applications cannot access them (nor can the user). When the user uninstalls your application, these files are removed.

Using Databases

- Most Android apps will need to persist user data at sometime.
- SQLite databases is a very convenient and speedy method of saving user (or app) data
- SQLite is a relational database management system just like Oracle, MySQL and PostgreSQL.
- SQLite is an opensource SQL database that stores the database as a text file on a device
- Due to its small footprint and public domain license, it is possibly the most widely deployed database engine in the world
- Android provides full support for **SQLite** databases.
- Any databases you create will be accessible by name to any class in the application, but not outside the application.

SQLiteOpenHelper

- The recommended method to create a new SQLite database is to create a subclass of SQLiteOpenHelper and override the onCreate() method
- You can then get an instance of our SQLiteOpenHelper implementation using the constructor you've defined.
- To write to and read from the database, call getWritableDatabase() and getReadableDatabase(), respectively.
- These both return a SQLiteDatabase object that represents the database and provides methods for SQLite operations.
- Every SQLite query will return a Cursor that points to all the rows found by the query.

StudentDBSQLite

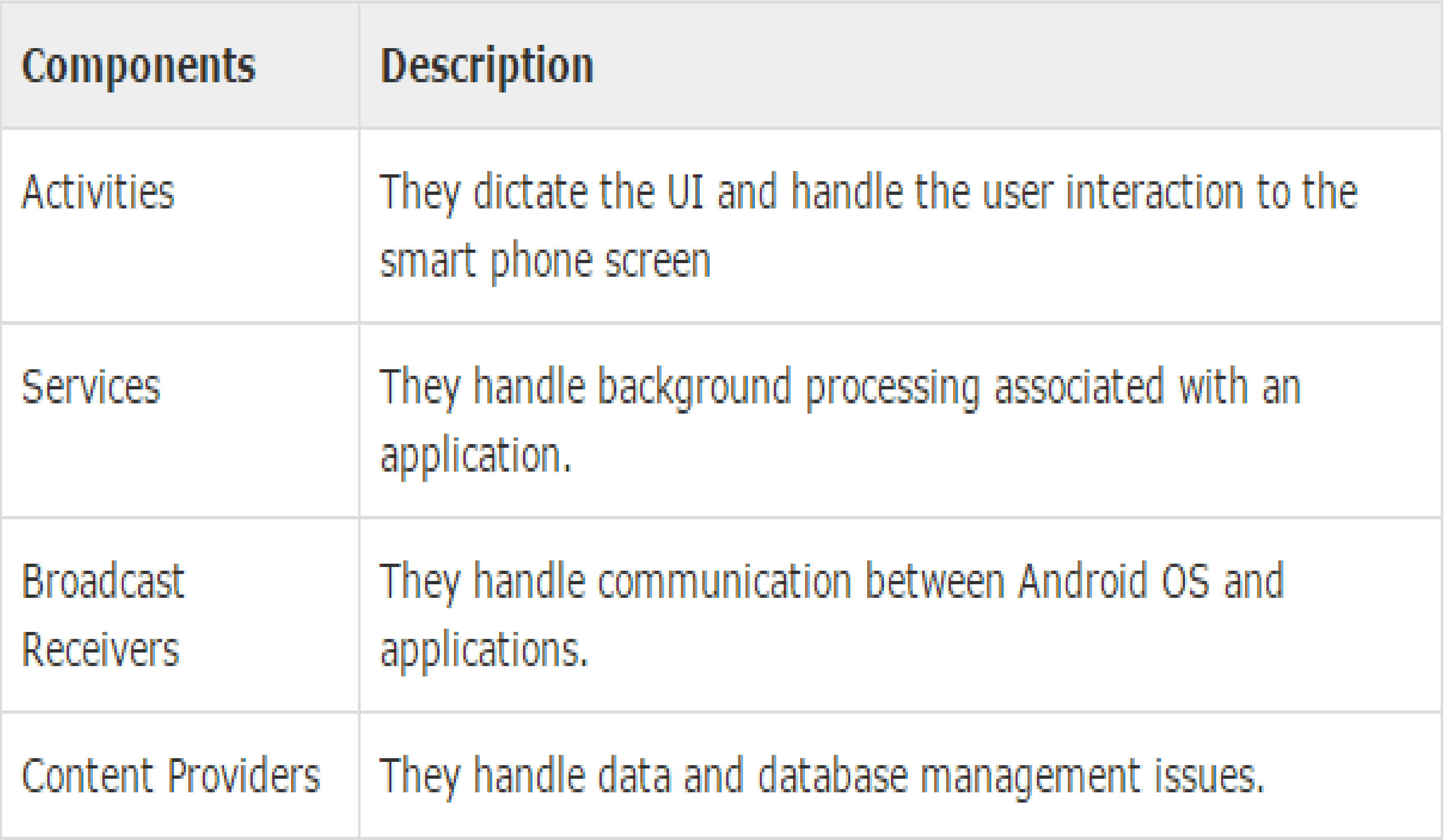
- The most important class, in our sample, is the **StudentDBSQLite** class, which extends **SQLiteOpenHelper**.
- SQLiteOpenHelper is a helper class designed to manage database creation and version management.
- You override onCreate() and onUpgrade() methods, and whenever a new database is created or upgraded, the appropriate method gets invoked.
- **StudentDBSQLite** is where all the SQLite operations are carried out, and both MainActivity and CreateStudent Activity call methods from this class to view, create, update or delete data from the database.

Preparation to use SQLite in an application

- The sample application in code handout (**DatabaseApp**) shows how to create a database, named “SQLiteStudentDB”, that contains a single table named “Student”. This table stores Student data, including his name, gender and age.
- There are two Activities, **MainActivity**, which shows a list of stored Student names, **CreateStudentActivity** , which allows adding Student details.

Application Components

- Application components are the essential building blocks of an Android application. These components are loosely coupled by the application manifest file *AndroidManifest.xml* that describes each component of the application and how they interact.



Components	Description
Activities	They dictate the UI and handle the user interaction to the smart phone screen
Services	They handle background processing associated with an application.
Broadcast Receivers	They handle communication between Android OS and applications.
Content Providers	They handle data and database management issues.

Activities

- An activity represents a single screen with a user interface,in-short Activity performs actions on the screen.

```
public_class_MainActivity_extends_Activity_{
}
```

- **Services**
- A service is a component that runs in the background to perform long-running operations. For example, a service might play music in the background while the user is in a different application, or **it might fetch data over the network without blocking user interaction with an activity.**

A service is implemented as a subclass of **Service** class as follows −

```
public_class_MyService_extends_Service_{
}
```

Broadcast Receivers

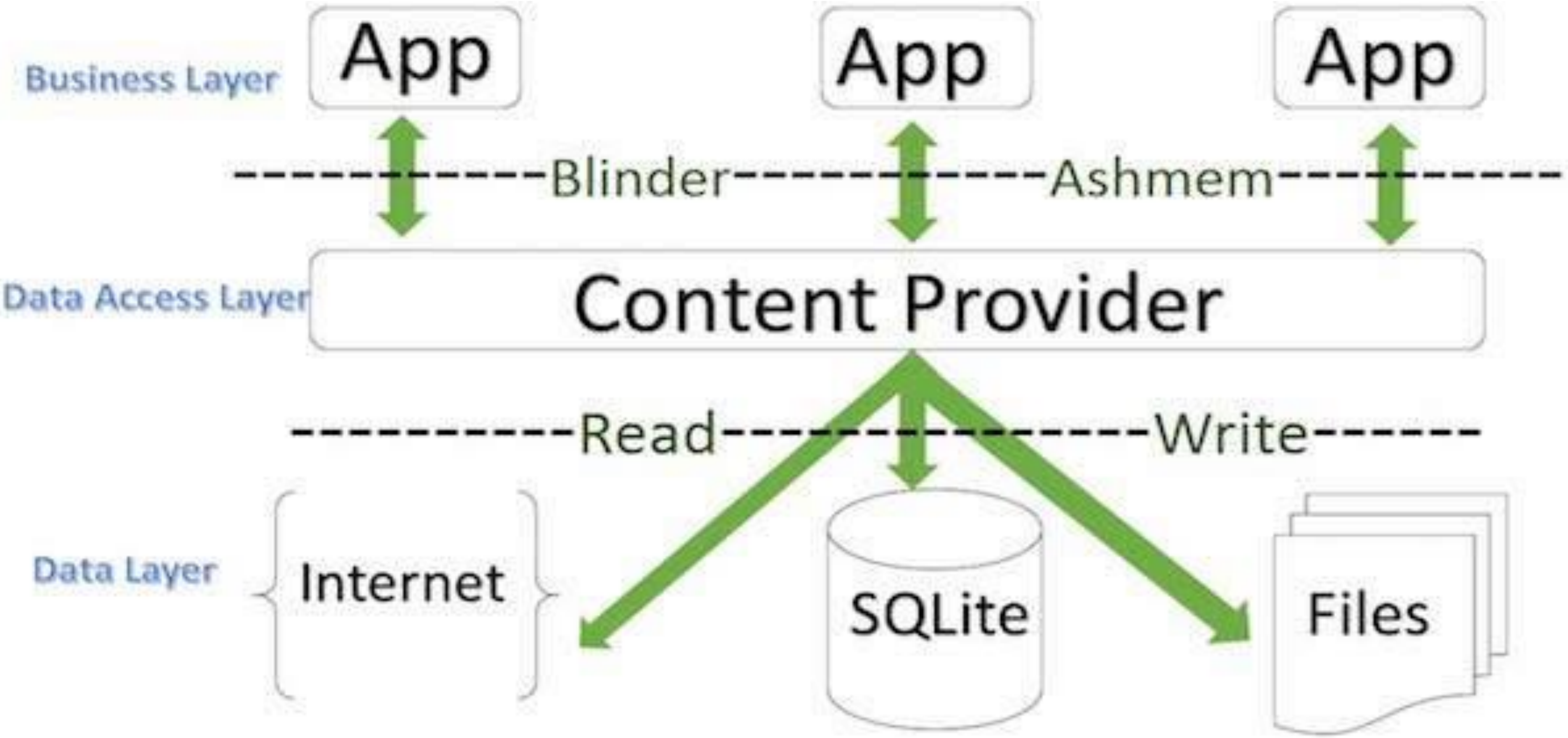
- Broadcast Receivers simply respond to broadcast messages from other applications or from the system. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use,

A broadcast receiver is implemented as a subclass of **BroadcastReceiver** class and each message is broadcaster as an **Intent** object.

```
public_class_MyReceiver_extends_BroadcastReceiver_{
__public_void_onReceive(context,intent){}
}
```

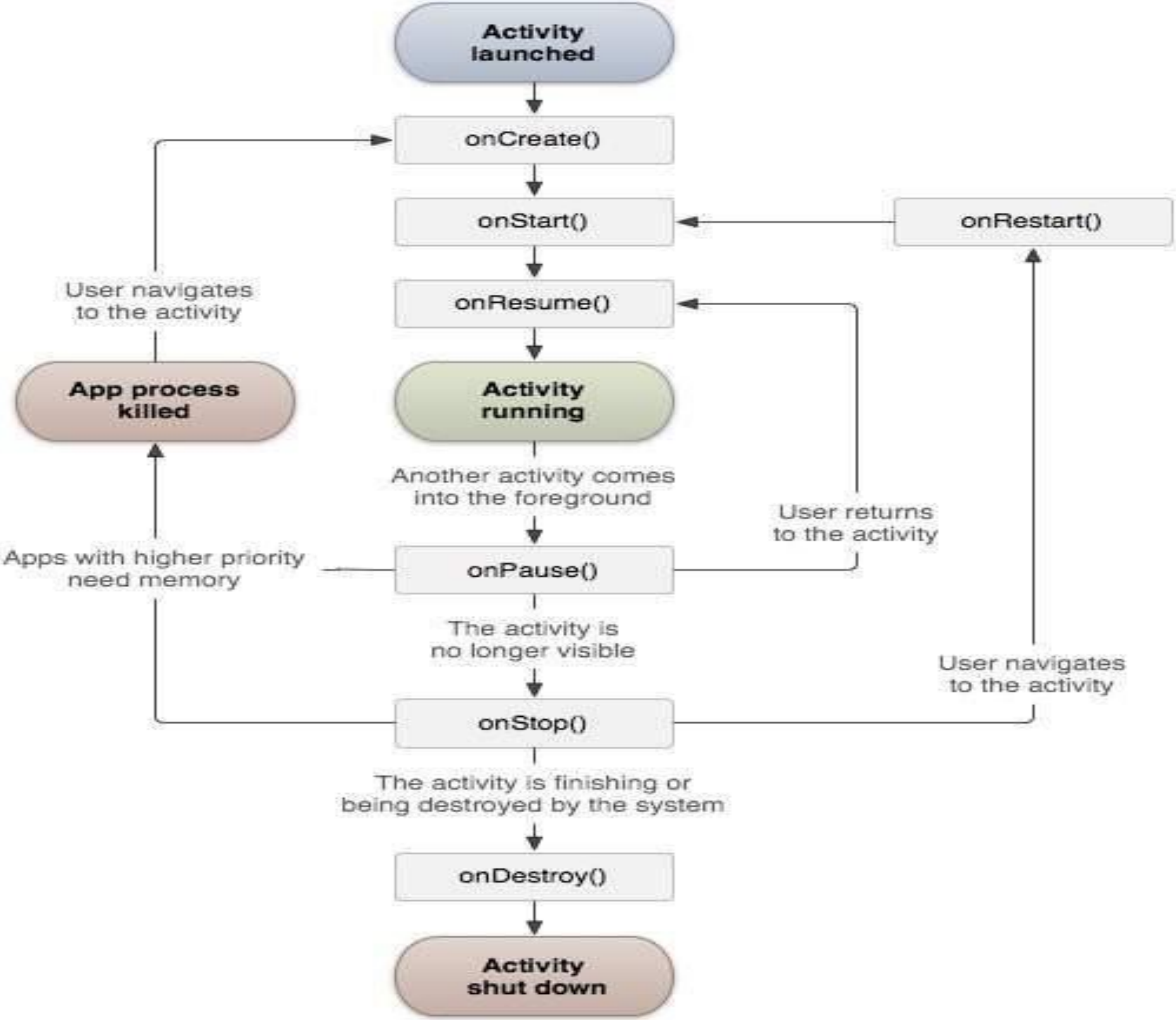
Content Providers

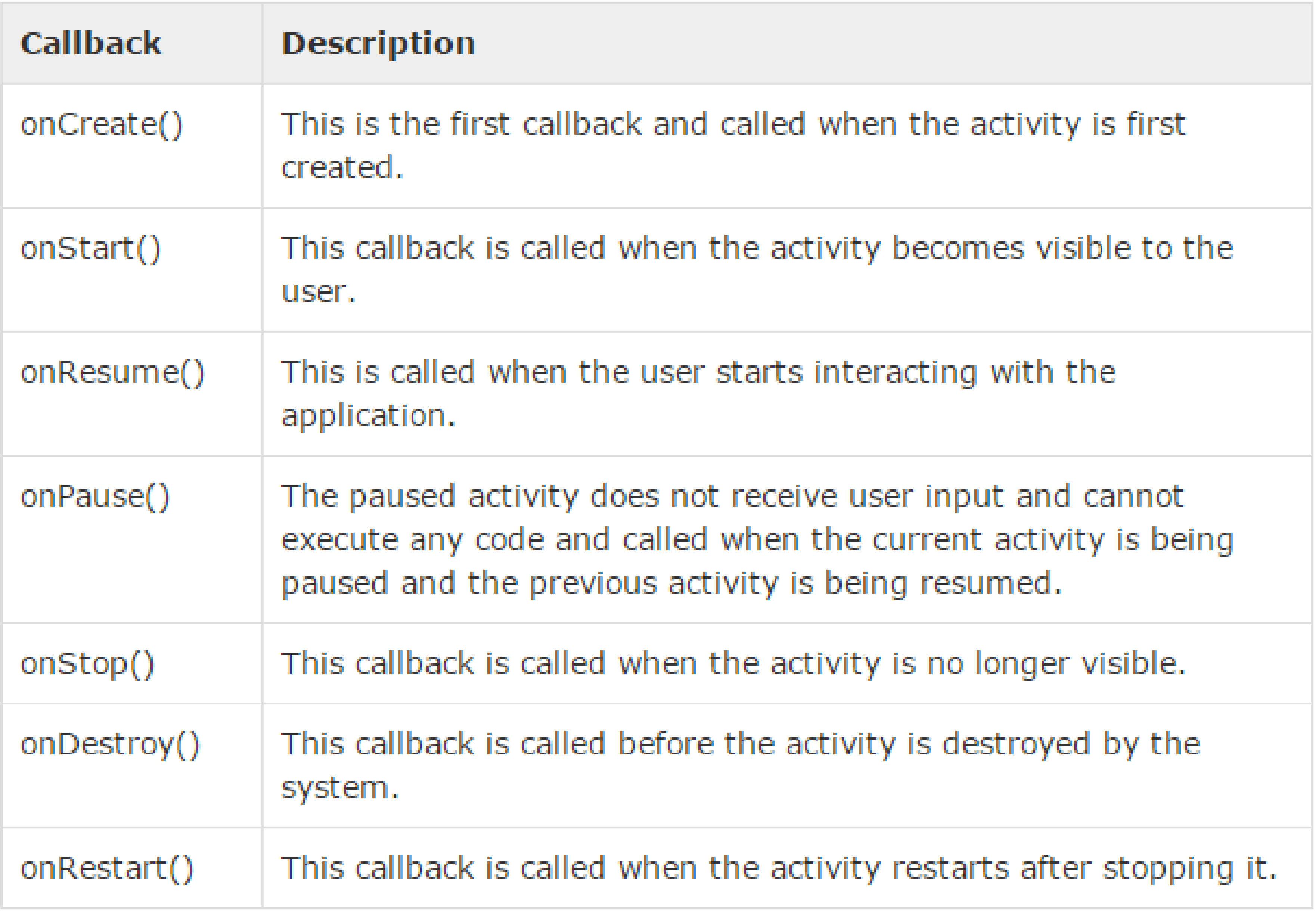
- A content provider component supplies data from one application to others on request.
- Such requests are handled by the methods of the *ContentResolver*class. The data may be stored in the file system, the database or somewhere else entirely. A content provider can use different ways to store its data and the data can be stored in a database, in files, or even over a network.


Business Layer
App
App
App
Blinder
Ashmem
Data Access Layer
Content Provider
Read
Write
Data Layer
Internet
SQLite
Files

Android Lifecycle

- If you have worked with C, C++ or Java programming language then you must have seen that your program starts from **main()** function. Very similar
- Android system initiates its program within an **Activity** starting with a call **on*onCreate()*** callback method.
- There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity as shown in the below Activity life cycle diagram


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down



Callback	Description
onCreate()	This is the first callback and called when the activity is first created.
onStart()	This callback is called when the activity becomes visible to the user.
onResume()	This is called when the user starts interacting with the application.
onPause()	The paused activity does not receive user input and cannot execute any code and called when the current activity is being paused and the previous activity is being resumed.
onStop()	This callback is called when the activity is no longer visible.
onDestroy()	This callback is called before the activity is destroyed by the system.
onRestart()	This callback is called when the activity restarts after stopping it.

Intent Objects

- An Intent object is a bundle of information which is used by the component that receives the intent as well as information used by the Android system.
- An Intent is a messaging object you can use to request an action from another app component.
- An Intent object can contain the following components based on what it is communicating or going to perform

Action

- This is mandatory part of the Intent object and is a string naming the action to be performed
- The action largely determines how the rest of the intent object is structured . The Intent class defines a number of action constants corresponding to different intents.
- The action in an Intent object can be set by the setAction() method and read by getAction().

Action

- **ACTION_VIEW** ***content://contacts/people/1*** -- Display information about the person whose identifier is "1".
- **ACTION_DIAL** ***content://contacts/people/1*** -- Display the phone dialer with the person filled in.
- **ACTION_WEB_SEARCH**

Data and Category

- Data adds a data specification to an intent filter.

- The category is an optional part of Intent object and it's a string containing additional information about the kind of component that should handle the intent.
  - For example, CATEGORY_LAUNCHER means it should appear in the Launcher as a top-level application, while CATEGORY_ALTERNATIVE means it should be included in a list of alternative actions the user can perform on a piece of data.

Extras Flags

- **Extras**: This will be in key-value pairs for additional information that should be delivered to the component handling the intent. The extras can be set and read using the putExtras() and getExtras() methods respectively.
- **Flags:** These flags are optional part of Intent object and instruct the Android system how to launch an activity, and how to treat it after it's launched etc.

Intent Filters

- You have seen how an Intent has been used to call an another activity. Android OS uses filters to pinpoint the set of Activities, Services, and Broadcast receivers that can handle the Intent with help of specified set of action, categories, data scheme associated with an Intent. You will use **<intent-filter>** element in the manifest file to list down actions, categories and data types associated with any activity, service, or broadcast

Intent Filters in Manifest

```
<activity android:name=".CustomActivity"
  android:label="@string/app_name">
  <intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <action android:name="com.example.My
   Application.LAUNCH" />
    <category
   android:name="android.intent.category.DEFAULT" />
    <data android:scheme="http" />
  </intent-filter>
</activity>
```

Intents usage

- Although intents facilitate communication between components in several ways, there are three fundamental use cases:
- Starting an activity
- Starting a service
- Delivering a broadcast

Intent types

- **Explicit intents** specify the component to start by name (the fully-qualified class name). You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start. For example, you can start a new activity in response to a user action or start a service to download a file in the background.
- **Implicit intents** do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it.
- Implicit Intent in Android can invoke other application in the device. We can open a URL in a browser or can make a call. Many other tasks can be performed. Intent has different action that is used with implicit intent. The actions are like Intent.ACTION_VIEW, Intent.ACTION_DIAL, Intent.ACTION_CALL etc

Example explicit intent

-
  ```
  Intent intent = new Intent(this, DownloadService.class);
   intent.setData(Uri.parse(fileUrl));
  startService(intent );
  ```

Example implicit intent

-
  ```
  Intent intent =  new Intent(Intent.ACTION_VIEW,
  Uri.parse("http://www.concretepage.com"));
  startService(intent );
  ```

-
  ```
  // Create the text message with a string
  Intent sendIntent = new Intent();
  sendIntent.setAction(Intent.ACTION_SEND);
  sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
  sendIntent.setType("text/plain");

  // Verify that the intent will resolve to an activity
  if (sendIntent.resolveActivity(getPackageManager()) != null) {
     startActivity(sendIntent);
  }
  ```

```
public void onClick(View v) {
 Intent i = new Intent(android.content.Intent.ACTION_VIEW,
   Uri.parse("http://www.example.com"));
       startActivity(i);
     }
```

Network Connection

- Android lets your application connect to the internet or any other local network and allows you to perform network operations.

Checking Network Connection

- Before you perform any network operations, you must first check that are you connected to that network or internet e.t.c.
  - For this android provides **ConnectivityManager** class. You need to instantiate an object of this class by calling **getSystemService()** method.

```
ConnectivityManager check = (ConnectivityManager)
this.context.getSystemService(Context.CONNECTIVITY_SERVICE);
```

- Once you instantiate the object of ConnectivityManager class, you can use **getAllNetworkInfo** method to get the information of all the networks. This method returns an array of **NetworkInfo**. So you have to receive it like this.

```
NetworkInfo[] info = check.getAllNetworkInfo();
```

Connected State

- You can check **Connected State** of the network. Its syntax is given below

```
for (int i = 0; i<info.length; i++){
   if (info[i].getState() ==
    NetworkInfo.State.CONNECTED){
      Toast.makeText(context, "Internet is connected
      Toast.LENGTH_SHORT).show();
   }
}
```

Performing Network Operations

- After checking that you are connected to the internet, you can perform any network operation. Here we are fetching the html of a website from a url.
- Android provides **HttpURLConnection** and **URL** class to handle these operations. You need to instantiate an object of URL class by providing the link of website. Its syntax is as follows –

```
String link = "http://www.google.com";
URL url = new URL(link);
```

uses-permission

- Note that to perform the network operations described in this lesson, your application **manifest** must include the following permissions:

```
<uses-permission android:name="android.permission.INTERNET" />
    <uses-permission
    android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Connect and Download Data

- You can use **HttpURLConnection** to perform a GET and download your data.

- In the following snippet, the **doInBackground()** method calls the method **downloadUrl**().
- The **downloadUrl**() method takes the given URL and uses it to connect to the network via **HttpURLConnection**.
- Once a connection has been established, the app uses the method getInputStream() to retrieve the data as an **InputStream**.

- // Given a URL, establishes an HttpUrlConnection and retrieves // the web page content as a InputStream, which it returns as // a string.
-

```
private String downloadUrl(String myurl) throws IOException {
    InputStream is = null;
    // Only display the first 500 characters of the retrieve // web page content.
    int len = 500;
```

-

```
    try {
        URL url = new URL(myurl);
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setReadTimeout(10000 /* milliseconds */);
        conn.setConnectTimeout(15000 /* milliseconds */);
        conn.setRequestMethod("GET");
        conn.setDoInput(true);
        // Starts the query
        conn.connect();
        int response = conn.getResponseCode();
        Log.d(DEBUG_TAG, "The response is: " + response);
        is = conn.getInputStream();

        // Convert the InputStream into a string
        String contentAsString = readIt(is, len);
        return contentAsString;

    // Makes sure that the InputStream is closed after the app is  // finished using it.
    } finally {
        if (is != null) {
            is.close();
        }
```

Getting JSON DATA

Just like you get other content from network, Json data follow same steps.

Create a java class called **JsonParser –** tJsonParser class has a method **makeServiceCall,** which will help to establish a connection to the remote content provider (php server which provide JSON Data).

The makeServiceCall takes 3 parameters the url – the address to the remote server, the method you connect with (post or get) and the list of parameters to send out.

Transmitting Network Data Using Volley

- Volley is an HTTP library that makes networking for Android apps easier and most importantly, faster.