Unit 9:

Android Elements – `Intent`

Services and Fragment

Outline

- Major Elements for Building Apps
  - Activities
  - Intents
- Services in Android
  - Foreground
  - Background
  - Bound
- Fragments
  - Fragment Lifecycle

Major Elements for Building Apps (Android)

- To construct an Android app, there are various components and associated mechanisms that can be used to allow them work together as a cohesive app, including:
  - Android Activities
  - Intents, Intent Filters
  - Broadcast Intents, Broadcast Receivers
  - Android Services
  - Content Providers
  - Application Manifest, Resources, Context
  - Fragments

Major Elements for Building Apps (Android – Activities)

- Android Activity
  - An activity is a single, standalone module of application functionality that usually correlates directly to a single user interface screen and its corresponding functionality; e.g.
    - An appointment application has an activity screen that displays appointments set up. The application might also utilize another activity screen where new appointments may be entered by the user
  - Android applications are mostly created with one or more activities

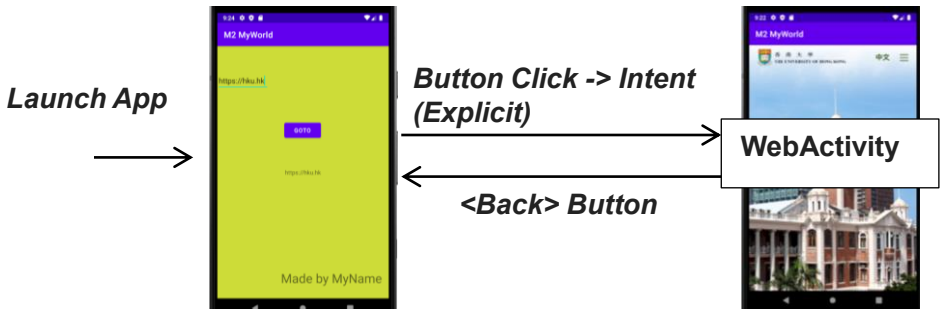
Major Elements for Building Apps (Android – Intents and Intent Filters)

- Intents
  - The mechanism by which one activity is able to launch another and implement the flow through the activities that make up an application
  - Intents consist of a description of the operation to be performed and, optionally, the performed data
  - Although intents facilitate communication between components in several ways, there are three fundamental use-cases: 1) start an activity, 2) start a service, and 3) deliver a broadcast
  - Two types of intents: *Explicit* and *Implicit* Intents

Major Elements for Building Apps (Android – Intents and Intent Filters)

- ***Explicit*** Intents specify which application will satisfy the intent, by supplying a component class name such as an activity class
  - Use an explicit intent to start a component in the own app, because we know the class name of the activity or service you want to start, e.g.
  - Sample below uses Explicit Intent to directly start another Java class `WebActivity`

```
// create an intent for starting a web activity below
Intent intent2Web = new Intent(this, WebActivity.class);
// start the intended activity
startActivity(intent2Web);
```



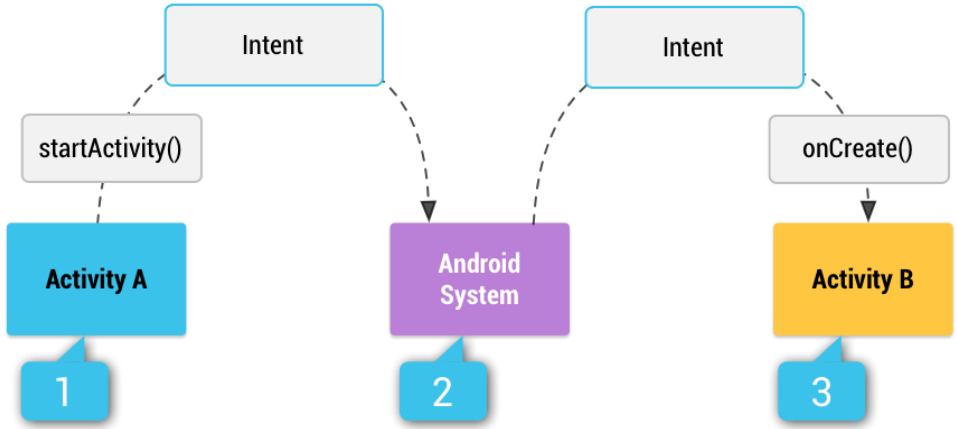
Major Elements for Building Apps (Android – Intents and Intent Filters)

- ***Implicit*** Intents specify type of action to be performed or data of a certain type to be handled, *e.g.*

```
startActivity( new Intent(Intent.ACTION_VIEW,
        Uri.parse("https://hku.hk")) );
```

  - Android runtime will select the activity to launch that most closely matches the criteria specified by this implicit Intent

Figure: How an implicit intent is delivered through the system to start another activity:
[1] Activity **A** creates an Intent with an action description and passes it to `startActivity()`.
[2] The Android System searches all apps for an intent filter that matches the intent.
[3] When a match is found, the system starts the matching activity (Activity **B**) by invoking its `onCreate()` method and passing it the Intent.



Major Elements for Building Apps (Android – Intents and Intent Filters)

- ***Implicit*** Intents and Intent Filters
  - To advertise which implicit intents an app can receive, declare intent filters for each component with an `<intent-filter>` element in manifest file
  - Each intent filter specifies the type it accepts based on intent's action, data, and category, *e.g.*

```
<!-- This activity handles "SEND" actions with text data -->
<intent-filter>
    <action android:name="android.intent.action.SEND"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="text/plain"/>
</intent-filter>
```

    - An intent filter to receive an ACTION_SEND intent when the data type is text
    - The system will deliver an implicit intent to the app component only if the intent can pass through one of intent filters

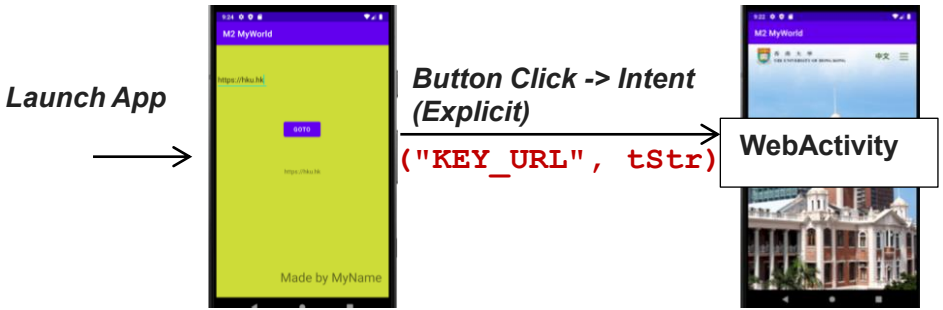
Major Elements for Building Apps (Android – Intents and Intent Filters)

- ***Put and get data*** with Intents
  - Before starting and sending an intent, we may attach/put it with some extra data, *e.g.*

```
// create an intent for a web activity below
Intent intent2Web = new Intent(this, WebActivity.class);
// add extra data to be passed, with key string "KEY_URL"
intent2Web.putExtra("KEY_URL", tStr); // tStr is a String
// start the intended activity
startActivity(intent2Web);
```

  - Correspondingly, we may get the put data back in intented class, *e.g.* in `WebActivity` above

```
// get the passed/put data(string) back, with same key string
String exStr = getIntent().getStringExtra("KEY_URL");
```



Major Elements for Building Apps (Android – Intents and Intent Filters)

- ***Receive result*** data back with Intents
  - May also start another activity with Intent, and receive a result back
  - E.g. we may have our own image editing app which starts another camera activity to take photo, and receive the photo result back to our own app
- To receive a result, start another activity by calling method `startActivityForResult()`, instead of `startActivity()`
- When the user is done with the subsequent activity and returns, the system calls the original activity's `onActivityResult()` method

```
protected void onActivityResult(int requestCode,
int resultCode, Intent data)
```

- We may also use the Activity Result APIs introduced in AndroidX Activity and Fragment

Major Elements for Building Apps (List of Sample Standard Actions for Intent)

Reference Only

- **ACTION_MAIN**
  - Activity Action: Start as a main entry point, does not expect to receive data.
  - Constant Value: **"android.intent.action.MAIN"**
- **ACTION_SEND**
  - Deliver some data to someone else. Value: **"android.intent.action.SEND"**
- **ACTION_VIEW**
  - Display the data to the user. This is the most common action performed on data -- it is the generic action we use on a piece of data to get the most reasonable thing to occur.
  - For example, when used on a contacts entry it will view the entry; when used on a mailto: URI it will bring up a compose window filled with the information supplied by the URI; when used with a tel: URI it will invoke the dialer.
- ACTION_CHOOSER
  - Display an activity chooser, allowing user to pick what they want to before proceeding.
- ACTION_DIAL
  - Dial a number as specified by the data.
- ACTION_WEB_SEARCH
  - Perform a web search.

Major Elements for Building Apps (List of Sample Standard Categories for Intent)

Reference Only

- **CATEGORY_LAUNCHER**
  - Should be displayed in the top-level launcher.
  - Constant Value: **"android.intent.category.LAUNCHER"**
- **CATEGORY_DEFAULT**
  - Set if the activity should be an option for the default action (center press) to perform on a piece of data.
  - Constant Value: **"android.intent.category.DEFAULT"**
- CATEGORY_BROWSABLE
  - Activities that can be safely invoked from a browser must support this category.
- CATEGORY_APP_BROWSER
  - Used with ACTION_MAIN to launch the browser application. The activity should be able to browse the Internet.
- CATEGORY_VR_HOME
  - An activity to use for the launcher when the device is placed in a VR Headset viewer. Used with ACTION_MAIN to launch an activity.

Major Elements for Building Apps (List of Sample Standard Broadcast Actions for Intent)

Reference Only

- **ACTION_TIME_TICK**
  - The current time has changed. Sent every minute.
  - Constant Value: **"android.intent.action.TIME_TICK"**
- ACTION_TIME_CHANGED
  - The time was set.
- ACTION_TIMEZONE_CHANGED
  - The timezone has changed.
- **ACTION_BOOT_COMPLETED**
  - This is broadcast once, after the user has finished booting.
- ACTION_POWER_CONNECTED
  - External power has been connected-to the device.
- ACTION_POWER_DISCONNECTED
  - External power has been disconnected-from the device.
- **ACTION_SHUTDOWN**
  - Device is shutting down.

Android Services (Overviews)

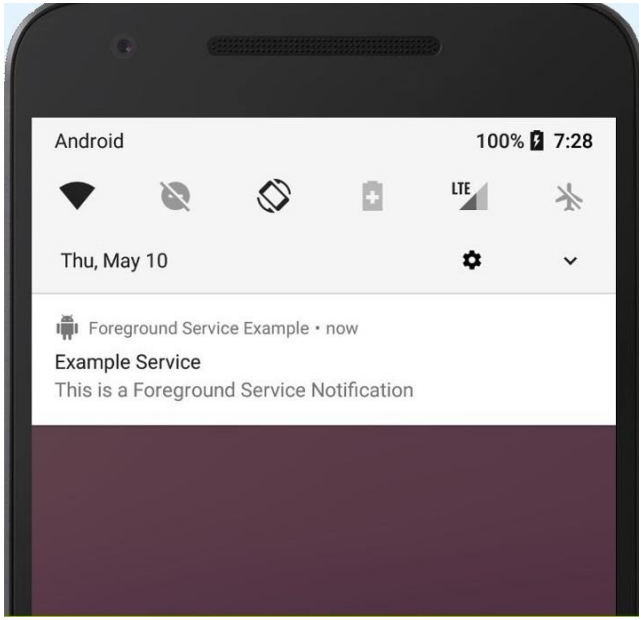
- A Service is an application component that can perform long-running operations in the background.
- No user interface is provided with no interaction between user and app.
- Once started, a service might continue running for some time, even after the user switches to another application.
- They are ideal for performing long-running operations, such as downloading files, syncing data, or processing data in the background.
- Additionally, a component can bind to a service to interact with it and even perform inter-process communication (IPC).
  - **IPC** is a mechanism which enables resource exchange and data sharing between the processes without interference.
- E.g. A service can handle network transactions, play music, perform file I/O, or interact with a content provider, all from the background.

Android Services (Overviews)

- In Android, Services play a crucial role in building robust and feature-rich applications.
  - Background Tasks and Long-Running Operations
  - Multitasking and User Experience Enhancement
- Users can multitask by switching between apps without interrupting ongoing background processes.
- E.g. music players often use services to continue playing music even when the user switches to a different app.
- A Service can be started and subsequently managed from activities, Broadcast Receivers or other Services.
- Android Services are ideal for situations where an app needs to continue performing tasks but does not necessarily need a user interface to be visible to the user.
- Although Services lack a user interface, they can still notify the user of events using notifications and are also able to issue Intents.

Android Services (Foreground Services)

- There are three types of services in Android·
- **Foreground Services**: These are services that perform some operation that is noticeable to the user, such as playing an audio track or downloading a file. Foreground services must display a notification to inform the user about the ongoing task. Foreground services continue running even when the user is not interacting with the app[1].
-



Android Services
(Background Services)

- These are services that perform an operation that is not directly noticed by the user, such as syncing data or compacting storage.
- Background services do not require a notification and can be stopped by the system when the app is not in the foreground·

Android Services
(Bound Services)

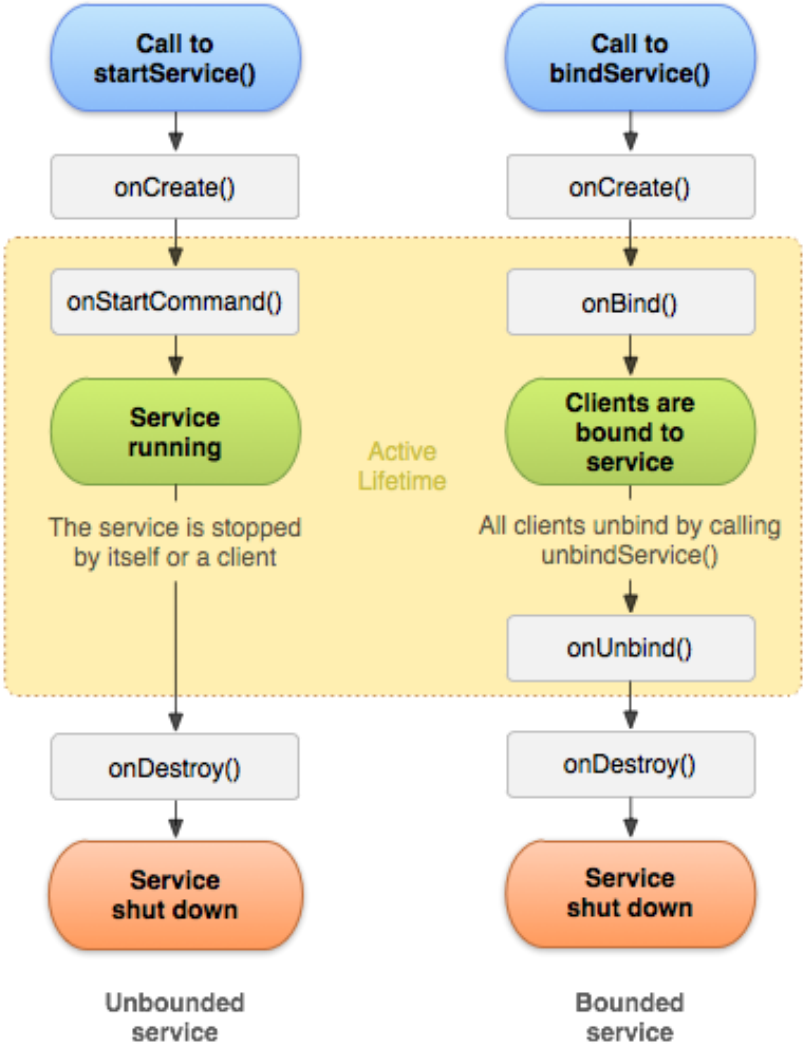
- These are services that allow other application components, such as activities, to bind to them and interact with them using inter-process communication (IPC).
- Bound services run only as long as another component is bound to them

Android Services Lifecycle Management:

- Services have two possible life cycle paths:
- Started Service (Unbounded Service):
  - Initiated using the `startService()` method. Continues running even if the component that started it is destroyed. Can be stopped using `stopService()` or `stopSelf()` methods.
- Bound Service:
  - Acts like a server in a client-server interface. Components can send requests to the service and fetch results. Bounded as long as any application component is bound to it.

Android Services Lifecycle Management:

- To create a service in Android, you need to follow these steps:
  - Declare the service in the manifest file using the <service> element.
  - Extend the Service class or one of its subclasses, such as IntentService or JobIntentService.
  - Implement the lifecycle methods of the service, such as `onCreate()`, `onStartCommand()`, `onBind()`, and `onDestroy()`.
  - Start the service from another component using `startService()` or `bindService()` methods.



Android Services
Example of creating a simple service

- Here is an example of creating a simple service that shows a toast message when it is started:

Here is an example of creating a simple service that shows a toast message when it is started:

```
// Declare the service in the manifest file
<manifest ... >

<application ... >

<service android:name=".ExampleService" />

</application>
</manifest>

// Extend the Service class
public class ExampleService extends Service {

// Implement the onCreate() method
@Override
public void onCreate() {
super.onCreate();
// Perform any initialization tasks here
}

// Implement the onStartCommand() method
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
// Perform any long-running tasks here
Toast.makeText(this, "Service started", Toast.LENGTH_SHORT).show();
return START_STICKY;
}
```

Android Services
Drawbacks

- Complexity and Debugging
  - Debugging issues related to services can be harder due to their separate process.
  - Memory leaks or resource management problems can occur if services are not handled correctly.
- Foreground Services Impact User Experience
  - Foreground services display notifications to the user, which can be intrusive. Users might find it annoying if an app keeps a persistent notification for a background task.
- Battery and Resource Consumption
  - Services running in the background can consume significant resources, including CPU cycles, memory, and battery.
- Foreground vs. Background Service Restrictions
  - Android imposes restrictions on background services from API 26. Background services may be killed by the system if the app is not in the foreground.

Android Services
Example of creating a simple service

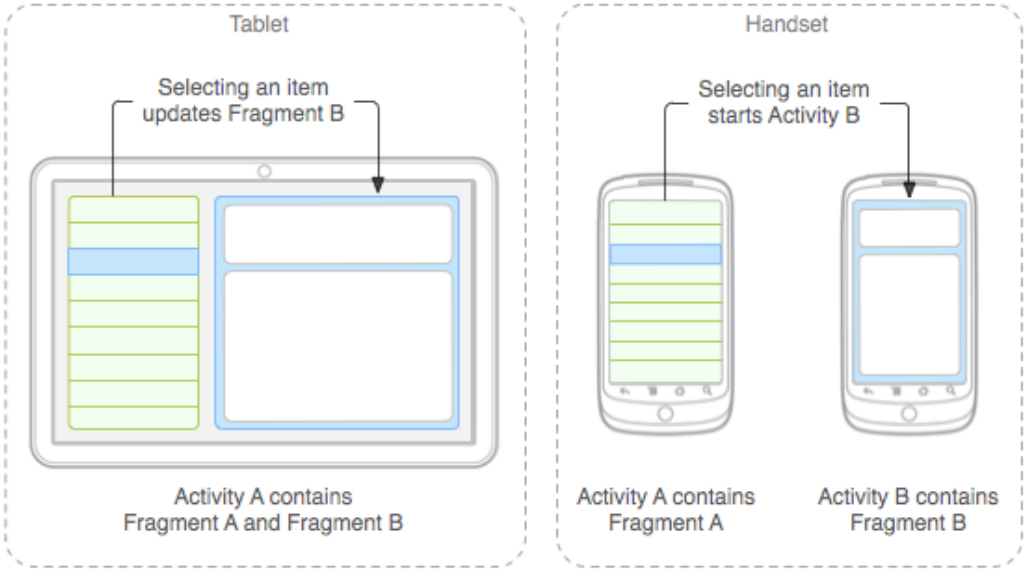
```
// Implement the onBind() method
@Override
public IBinder onBind(Intent intent) {
// Return null if the service is not bound
return null;
}

// Implement the onDestroy() method
@Override
public void onDestroy() {
super.onDestroy();
// Perform any cleanup tasks here
Toast.makeText(this, "Service stopped", Toast.LENGTH_SHORT).show()
}
}

// Start the service from another component
Intent intent = new Intent(this, ExampleService.class);
startService(intent);
```

Fragments

- A Fragment represents a reusable module of the app's User Interface.
- A fragment defines and manages its own layout, has its own lifecycle, and can handle (receives) its own input events.
- Fragments can't create on their own (independently) in the project's User Interface.
  - They must be hosted by an activity or another fragment.
- The fragment's view hierarchy becomes part of, or attaches to, the host's view hierarchy.

Fragments

- Developer can add or remove fragment while the activity is running (like a "sub-activity" that you can reuse in different activities)
- It represents a behavior or a portion of user interface in an Activity
- We can combine multiple fragments in a single activity to build a multi-pane (insulated) interface and reuse a fragment in multiple activities



Advantages of Fragments

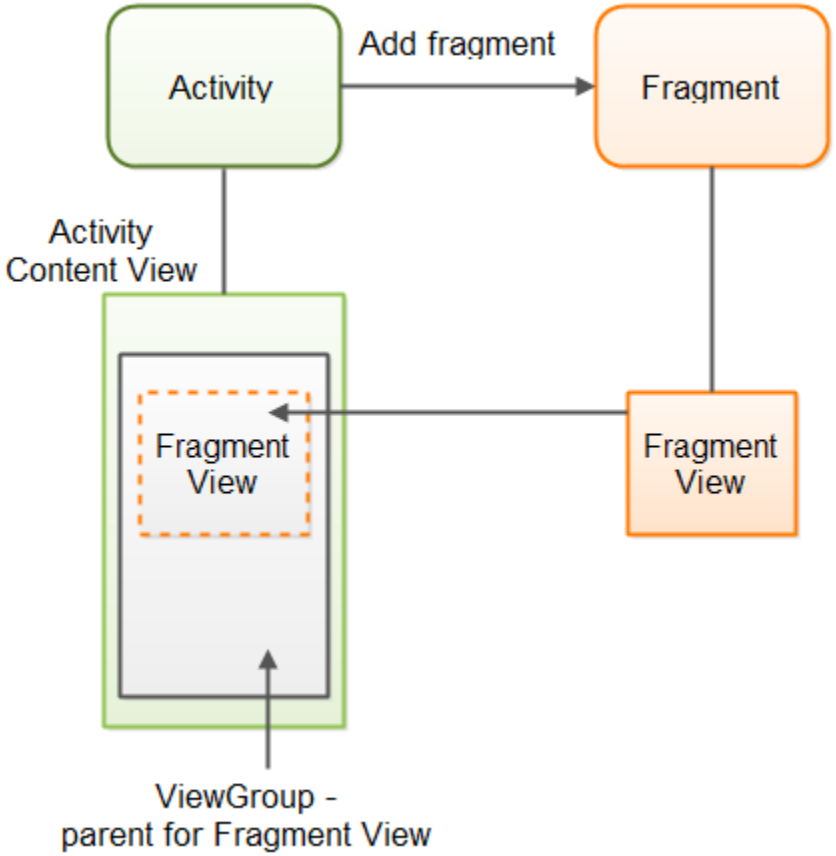
- A greatest advantage of fragments is that it simplifies the task of creating UI for multiple screen sizes. A activity can contain any number of fragments.

- An Android fragment is not by itself a subclass of View which most other UI 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).
  - A fragment is added to a ViewGroup inside the activity. The fragment's view is displayed inside this ViewGroup.

Advantages of Fragments

- The following diagram shows depicts what happens when a fragment is added to an activity:

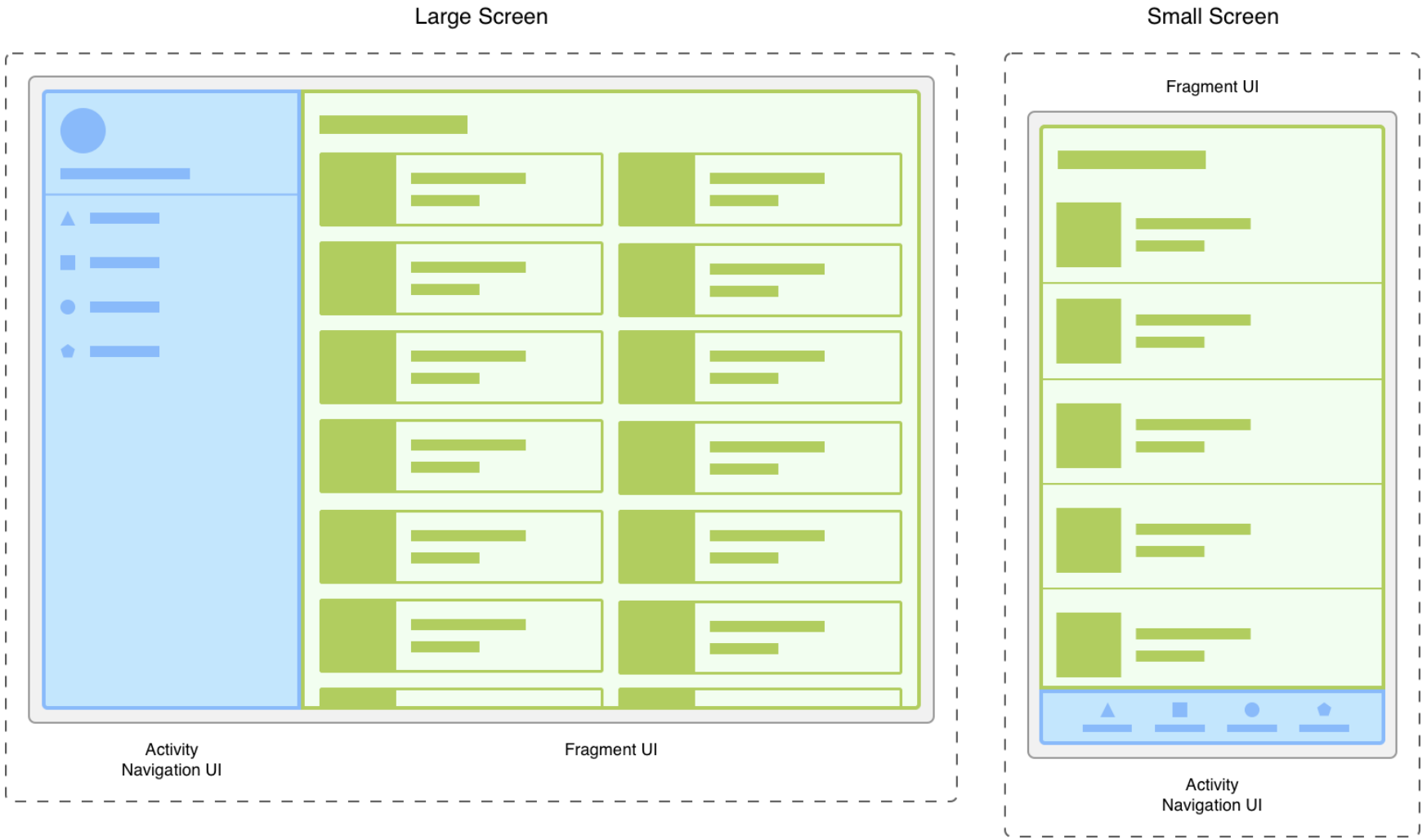


Advantages of Fragments

- Division of Screen with flexibility
  - On larger screens, you might want the app to display a static navigation drawer and a list in a grid layout.
  - On smaller screens, you might want the app to display a bottom navigation bar and a list in a linear layout.

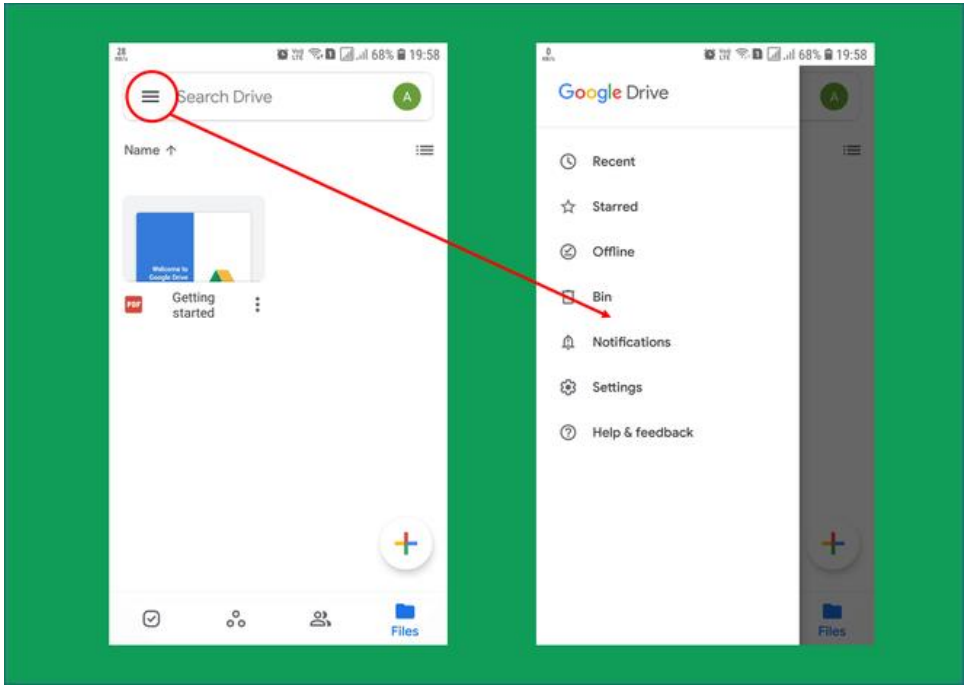
- Separating the navigation elements from the content can make this process more manageable.
  - The activity is then responsible for displaying the correct navigation UI, while the fragment displays the list with the proper layout.

Fragments


Large Screen
Small Screen
Fragment UI
Activity
Navigation UI
Fragment UI
Activity
Navigation UI

Fragments

- Modularity of Fragments
  - Fragments introduce modularity and reusability into the activity's interface by letting developer divide the interface into discrete chunks (layers).
  - Activities are an ideal place to put global elements around your app's user interface, such as a navigation drawer. Conversely, fragments are better suited to define and manage the UI of a single screen or portion of a screen.



Types of Android Fragments

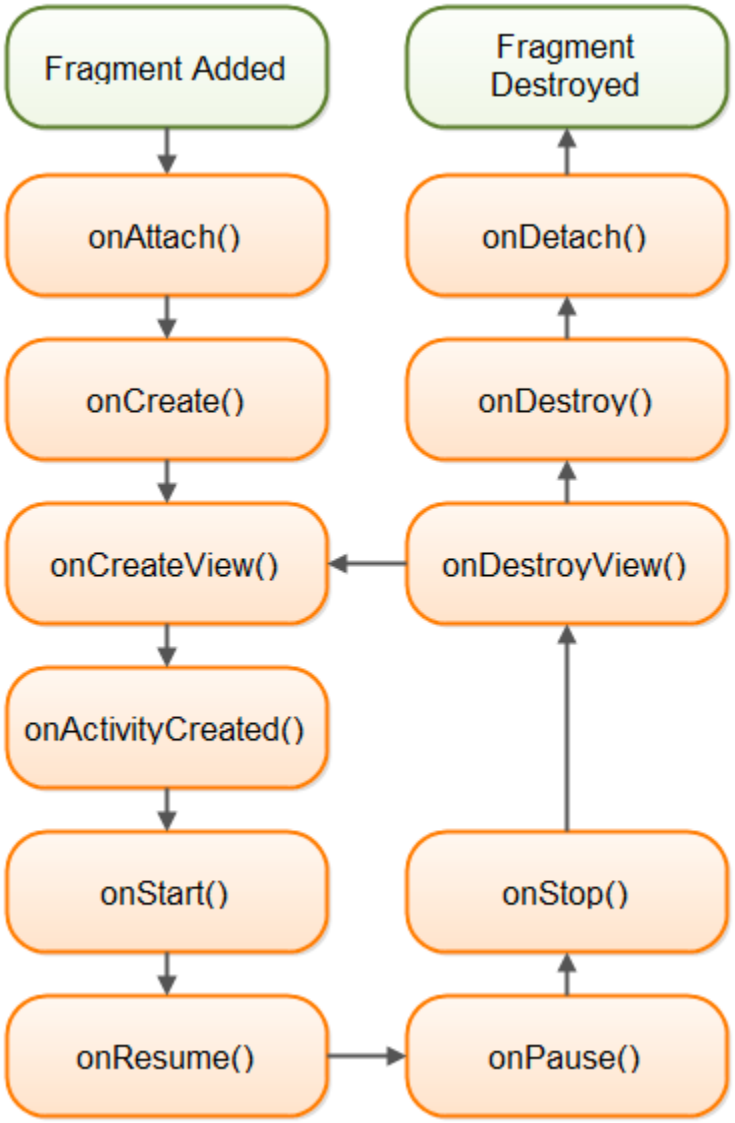
- Single Fragment:
  - Displays only one single view on the device screen. This type of fragment in android is mostly used for mobile phones.

- List Fragment:
  - Displays a list-view from which the user can select the desired sub-activity. The menu drawer of apps like Gmail is the best example of this kind of android fragment.

- Fragment Transaction:
  - Supports the transition from one fragment in android to another at run time. Users can switch between multiple fragments like switching tabs.

Fragment Life Cycle

- Fragments
  - A fragment must always be embedded in an activity and the fragment's lifecycle is directly affected by the host activity's lifecycle
    - For example, when the activity is paused, so are all fragments in it, and when the activity is destroyed, so are all fragments
  - However, while an activity is running (it is in the resumed lifecycle state), we can manipulate each fragment independently, such as add or remove them



Fragment Life Cycle

- The Fragment Lifecycle controls the fragment's existence, from creation to destruction.

- Key callbacks include:
  - onCreate() — Initializes the fragment. Called only once.
  - onStart() — When the fragment becomes visible.
  - onDestroy() — Cleans up resources when the fragment is permanently removed.

- A fragment can persist in memory even when its view is destroyed, for example when navigating away using FragmentTransaction.replace().

View Lifecycle for Fragment

- In contrast, the View Lifecycle for fragment manages the fragment's UI. Important callbacks include:
  - onCreateView() — Called to create the fragment's UI hierarchy.
  - onViewCreated() — Where you initialize views or set up data.
  - onDestroyView() — Cleans up the view hierarchy when the user navigates away from the fragment.

View Lifecycle for Fragment

- Below are the methods of fragment lifecycle.
  1. onAttach()
     - This method will be called first, even before onCreate(), letting us know that your fragment has been attached to an activity. You are passed the Activity that will host your fragment
  2. **onCreate()**
     - Initializes the fragment. Called only once.
  3. onCreateView() :
     - The system calls this callback when it's time for the fragment to draw its UI for the first time.
     - To draw a UI for the fragment, a View component must be returned from this method which is the root of the fragment's layout. We can return null if the fragment does not provide a UI
  4. onViewCreated() :
     - This will be called after onCreateView().
     - This is particularly useful when inheriting the onCreateView() implementation but we need to configure the resulting views, such as with a ListFragment and when to set up an adapter

View Lifecycle for Fragment

5. onActivityCreated() :
   - This will be called after onCreate() and onCreateView(), to indicate that the activity's onCreate() has completed. If there is something that's needed to be initialised in the fragment that depends upon the activity's onCreate() having completed its work then onActivityCreated() can be used for that initialisation work

6. **onStart() :**
   - The onStart() method is called once the fragment gets visible

6. onPause() :
   - The system calls this method as the first indication that the user is leaving the fragment. This is usually where you should commit any changes that should be persisted beyond the current user session

7. onStop() : Fragment going to be stopped by calling onStop()

View Lifecycle for Fragment

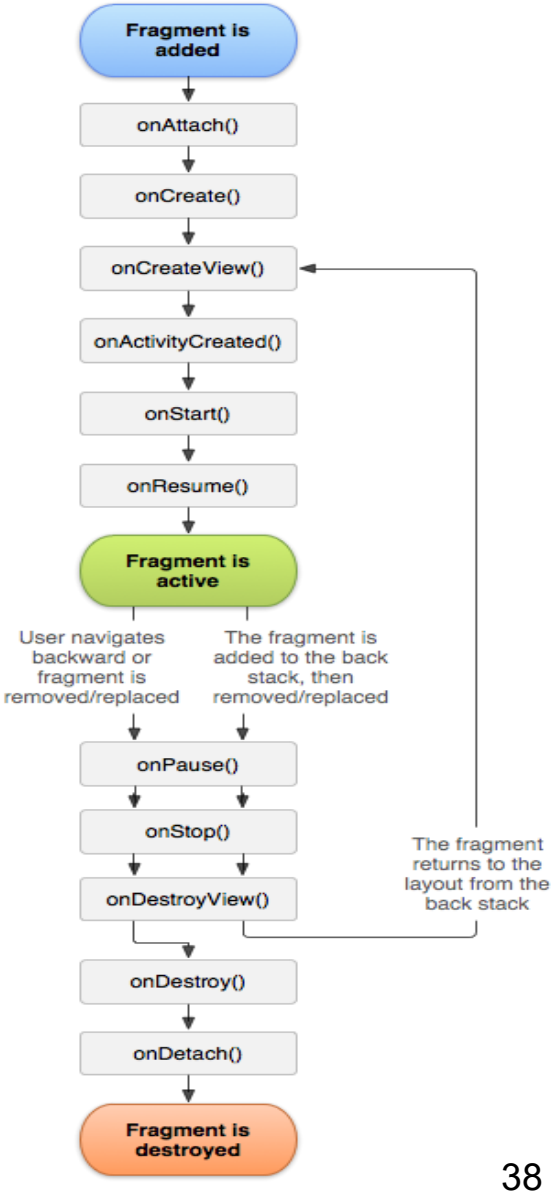
9. onDestroyView() :
   - It's called before onDestroy(). This is the counterpart to onCreateView() where we set up the UI. If there are things that are needed to be cleaned up specific to the UI, then that logic can be put up in onDestroyView()

10. **onDestroy() :**
    - onDestroy() called to do final clean up of the fragment's state but Not guaranteed to be called by the Android platform.

11. onDetach() :
    - It's called after onDestroy(), to notify that the fragment has been disassociated from its hosting activity

Major Elements for Building Apps (Fragment Life Cycle)

- Fragment Lifecycle
  - Managing the lifecycle of a fragment is a lot like managing the lifecycle of an activity
  - Like an activity, a fragment can exist in three states:
    - Resumed: The fragment is visible in the running activity.
    - Paused: the activity in which this fragment lives is not on focus, but still visible
    - Stopped: The fragment is not visible

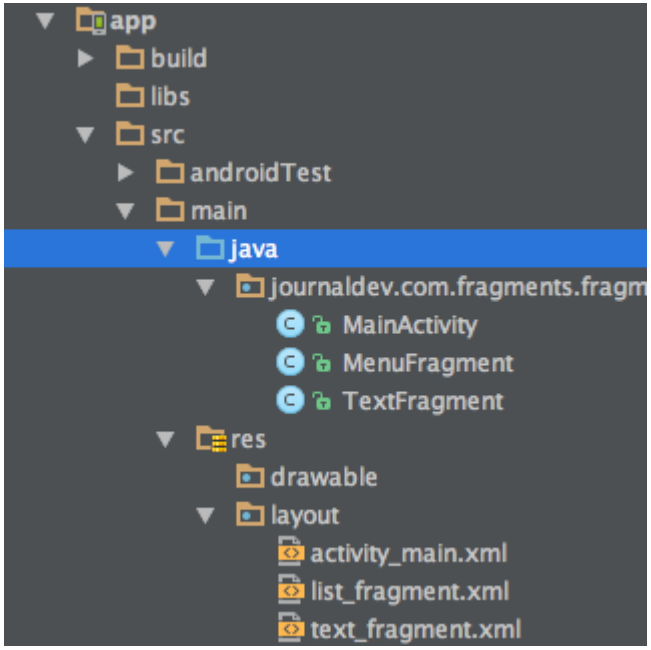


Android Fragment Classes

- Fragments were added to the Android API in Honeycomb(API 11).
  - android.app.Fragment :
    - The base class for all fragment definitions
  - android.app.FragmentManager :
    - The class for interacting with fragment objects inside an activity
  - android.app.FragmentTransaction :
    - The class for performing an atomic set of fragment operations When using a compatibility package library provided by Google, the following classes are used for implementation.

Android Fragment Classes

- Android fragments example project comprises of a single activity holding two fragments: TextFragment and MenuFragment respectively.



Fragment inside Activity

- The MainActivity.xml below holds the two fragments TextFragment and MenuFragment. So lets begin with defining the fragments in the xml layout activity_main.xml

```
<LinearLayout xmlns:android="https://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
  <fragment
      android:layout_height="match_parent"
      android:layout_width="match_parent"
      class="journaldev.com.fragments.fragments.MenuFragment"
      android:id="@+id/fragment"
      android:layout_weight="0.5"/>
  <fragment
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      class="journaldev.com.fragments.fragments.TextFragment"
      android:id="@+id/fragment2"
      android:layout_weight="0.5"/>
</LinearLayout>
```

References

- Part of this slide set is prepared and extracted from the followings:
  - Gok, N. and Khanna, N. (2013) *'Building Hybrid Android Apps with Java and JavaScript'*, O'Reilly Media, Inc.
  - Gargenta, M. and Nakamura, M. (2014) *'Learning Android'*, 2ed, O'Reilly Media, Inc.
  - Google Inc. 'Website for Android Developers', http://developer.android.com/
  - Google Inc. 'Website for Android Sources', https://source.android.com/
  - Apple Inc. 'Website for Apple Developers', Website: https://developer.apple.com/
  - Microsoft Corp. 'Website for Windows Developers', https://developer.microsoft.com/en-us/windows
  - Wikipedia Website: http://en.wikipedia.org
  - https://developer.android.com/guide/fragments
- This set of slides is for teaching purpose only