CS683 Mobile Application Development

Advanced UI

Review

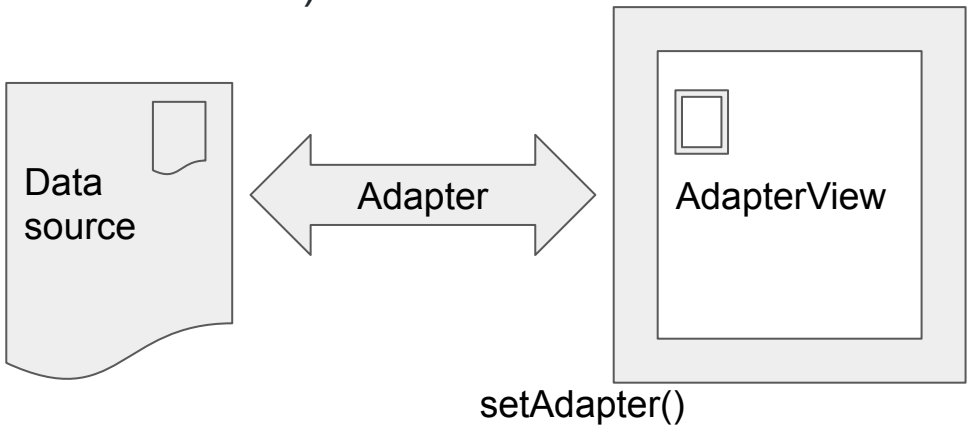
- Activity vs Fragment
- View vs. ViewGroup
- Event Listener
- Navigate between Activities (Intent)
- Navigate between Fragments (Navigation Component)

View Binding

- *View binding* allows you to more easily write code that interacts with views.
- Once view binding is enabled in a module, it generates a *binding class* for each XML layout file present in that module.
- An instance of a binding class contains direct references to all views that have an ID in the corresponding layout.
- View binding can replace findViewById.
- https://developer.android.com/topic/libraries/view-binding

AdapterView & Adapter

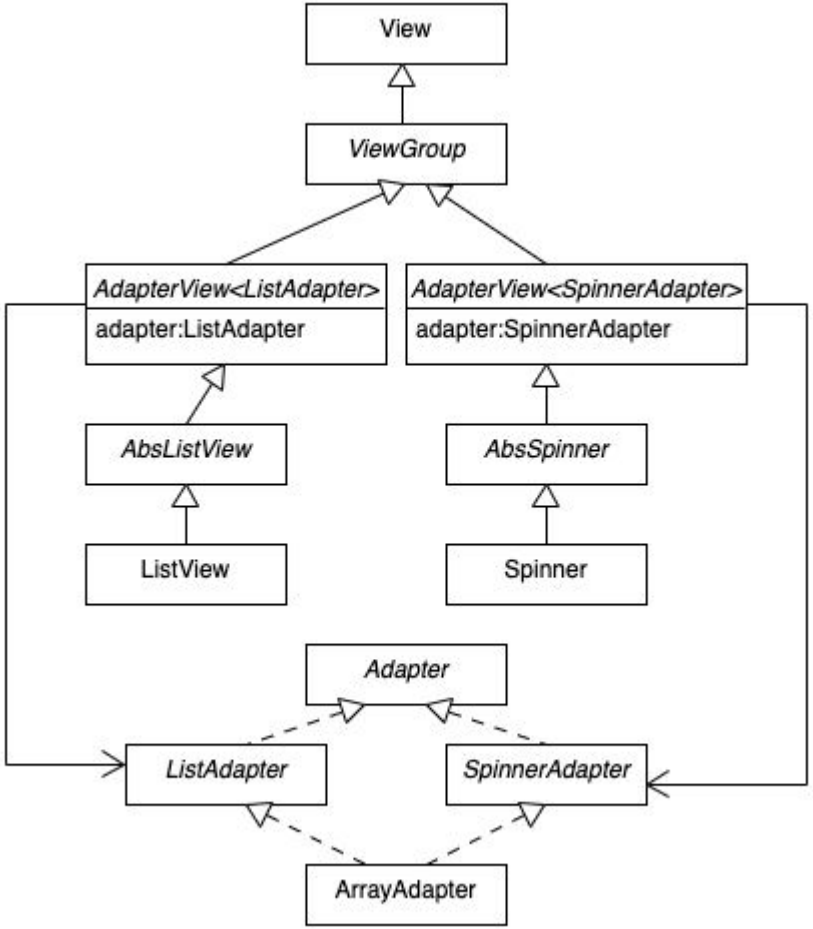
- An Adapter object acts as a bridge between an AdapterView and the underlying data for that view. The Adapter provides access to the data items. The Adapter is also responsible for making a View for each item in the data set.
- The adapter view also specifies how to display these children views (e.g the relative positions of these views).



AdapterView

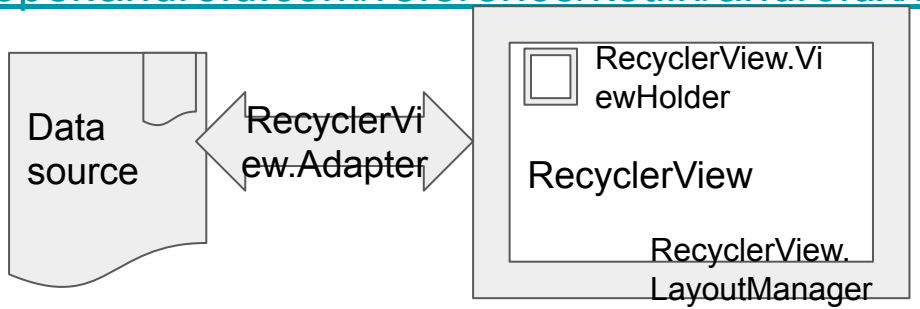
- ListView
  - It is an adapter view.
  - It displays a vertically-scrollable collection of views, where each view is positioned immediately below the previous view in the list.
  - A ListView can only work with a ListAdapter. An ArrayAdapter is also a ListAdapter.
- Gridview
- Spinner
  - To display a constant string, we can use android:entries property of the adapter view, for example, when working with spinners and listviews.
  - If the data will change, you need to create a customized adapter in the code to bind the data.
- https://developer.android.com/reference/android/widget/AdapterView

Adapter View



RecyclerView

- It is very similar to ListViews. However, it is NOT inherited from AdapterView. It is provided in the support library and now it is in the Androidx namespace (androidx.recyclerview.widget.RecyclerView).
- To use a recyclerview, we need first to subclass RecyclerView.Adapter to create an adapter for the Recyclerview. The subclass needs to implement three abstract methods. It also needs to specify the associated data.
- Use the setAdapter() method of the RecyclerView to set its adapter.
- https://developer.android.com/reference/kotlin/androidx/recyclerview/widget/RecyclerView



RecyclerView.Adapter

- It is the base class for all recyclerview adapters.
- It is an abstract class.
- A recylcerview adapter must implement three abstract method
  - getItemCount()
  - onBindViewHolder()
  - onCreateViewHolder()
  - A ViewHolder describes an item view and metadata about its place within the RecyclerView.

Communication between Fragments

- Fragments are not independent Android components. They depend on activities that contain them. They cannot communicate with each other directly. It is also not recommended for fragments to communicate with the hosting activities directly either.
- One way to communicate is to define destination arguments in the navigation graph.
- Another way is to use ViewModel.

Destination Arguments

- Only use this method to pass a very small amount of data between destinations.
- Navigation graph defines destinations and actions.
- Define arguments for fragments in the navigation graph
- Define arguments for actions in the navigation graph
- Receive arguments in the fragments.
- Create augmented actions for navigation.
- https://developer.android.com/guide/navigation/navigation-pass-data

MVC

- M - Model
- V - View
  - View, View group, XML Layout
- C - Controller
  - Activity, fragment (UI controller)

UI Controllers

- Problems of data in UI controllers (activities or fragments)
  - If the system destroys or re-creates a UI controller (e.g. screen rotations), any transient UI-related data you store in them is lost.
  - If the system destroys or re-creates a UI controller (e.g. screen rotations), the return of the asynchronous calls that the UI controller made may cause potential memory leaks.

ViewModel

- Separate the UI-related data handling from UI controller logic. Make UI controllers simpler
- Designed to store and manage UI-related data in a lifecycle conscious way.
  - Allow data to survive configuration changes such as screen rotations. Any data contained in the ViewModel will stay until its lifecycle owner finishes.
  - Prevent data leakage in the asynchronous call.
- https://developer.android.com/topic/libraries/architecture/viewmodel?gclid=CjwKCAjwloCSBhAeEiwA3hVo_cx-Cf3QDqsu7pmRDZwG0FFnOA9yUsvK1yvD6aA3u-797a6lSqwX9xoCci8QAvD_BwE&gclsrc=aw.ds

MVVM

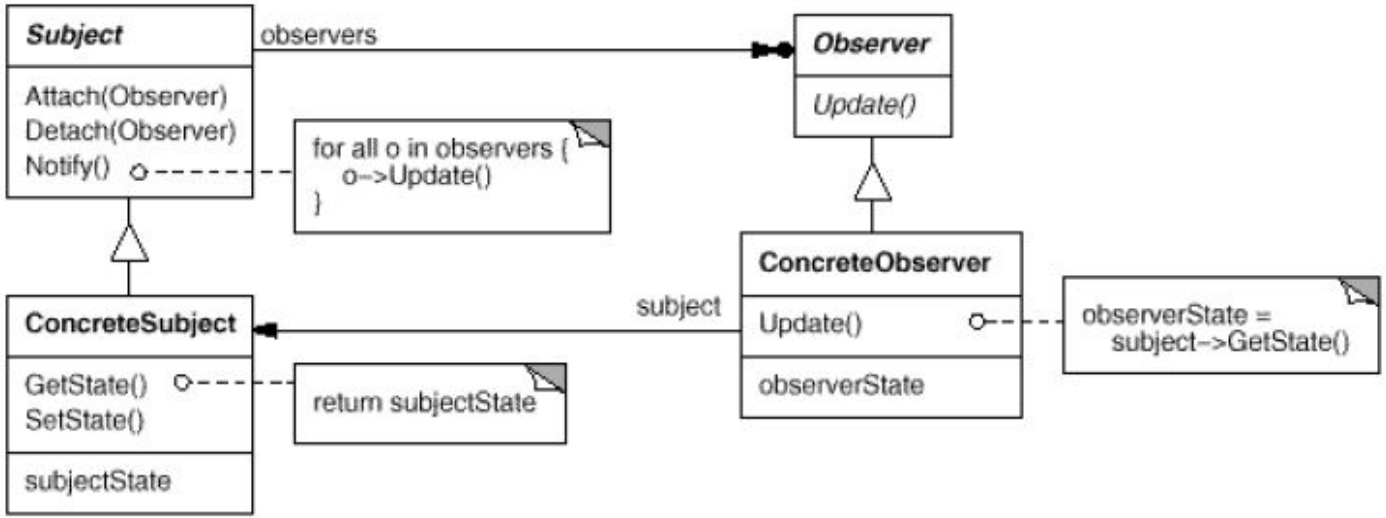
- View
  - UI components, UI controller
- ViewModel
  - UI-related data
- Model
  - Application data, business logic
  - Database, file, network data

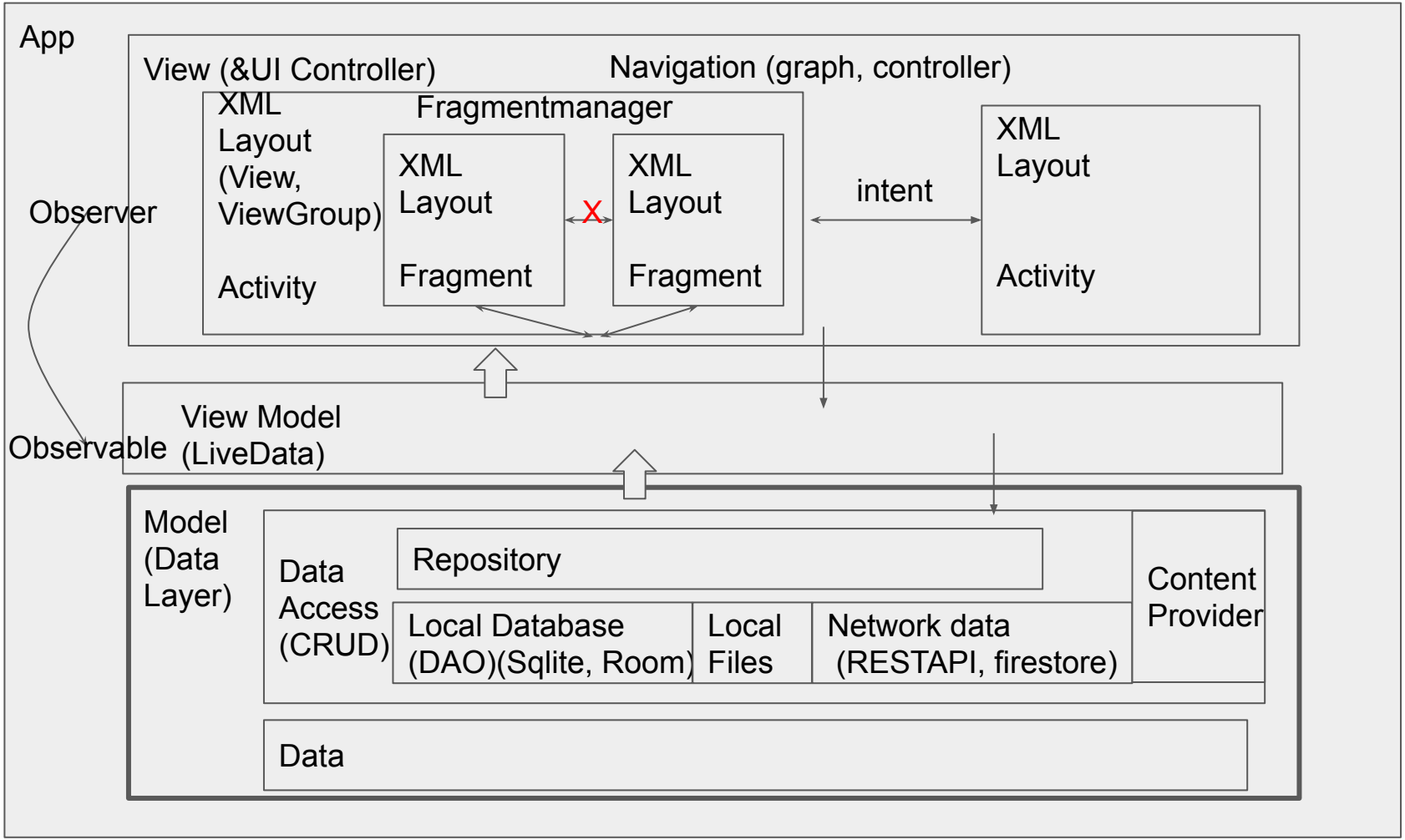
LiveData

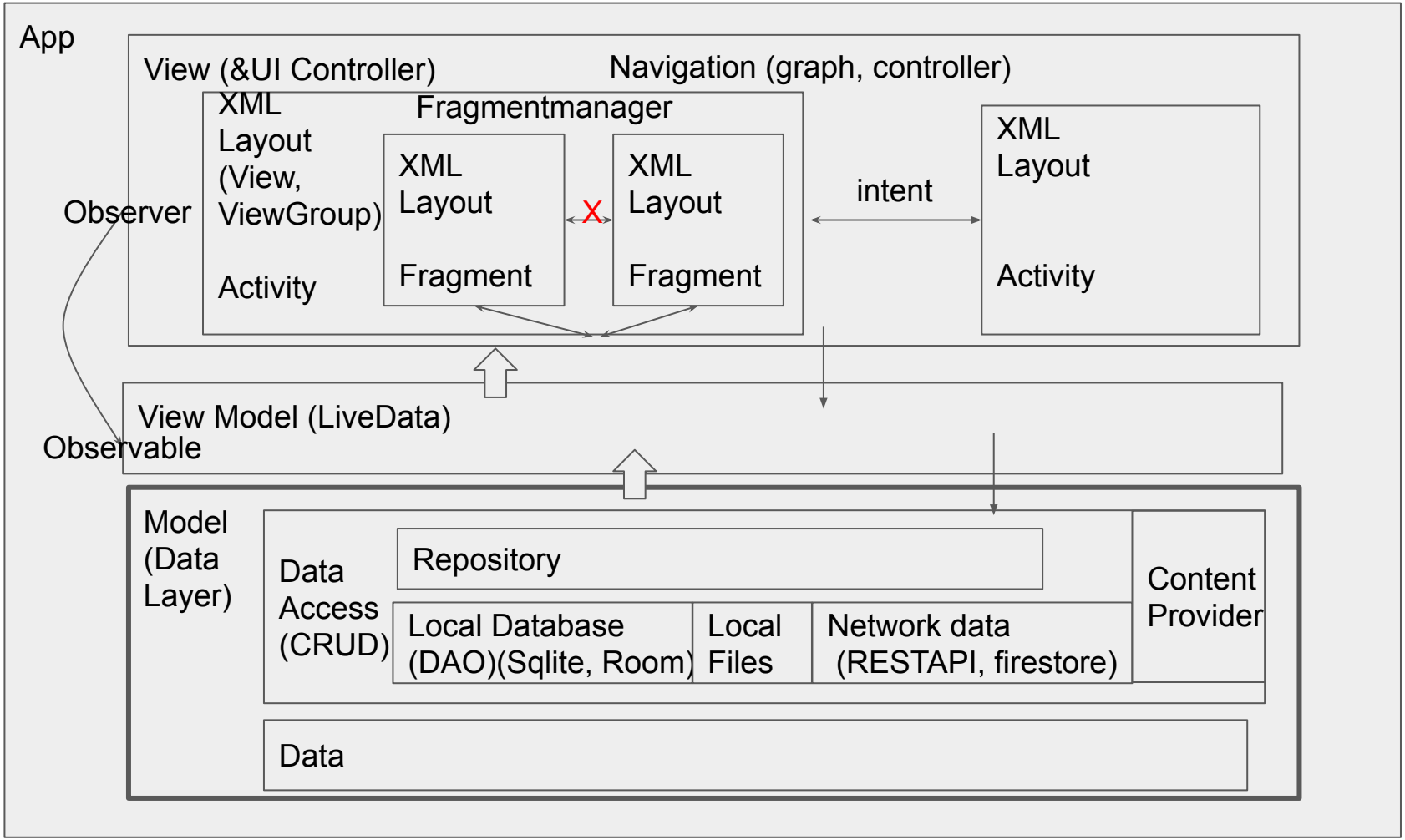
- LiveData, MutableLiveData
  - An observable data holder class.
  - Lifecycle-aware, only notifies active observers (e.g. in STARTED or RESUMED state) about updates.
  - MutableLiveData: liveData value that can be changed explicitly using setValue() or postValue()
  - https://developer.android.com/topic/libraries/architecture/livedata

Observer Pattern

- When the observable (subject) state changes, the associated observers will update automatically




App
View (&UI Controller)
Navigation (graph, controller)
XML
Layout
(View,
ViewGroup)
Activity
Fragmentmanager
XML
Layout
Fragment
X
XML
Layout
Fragment
intent
XML
Layout
Activity
Observer
Observable
View Model
(LiveData)
Model
(Data
Layer)
Data
Access
(CRUD)
Repository
Local Database
(DAO)(Sqlite, Room)
Local
Files
Network data
(RESTAPI, firestore)
Content
Provider
Data


App
View (&UI Controller)
Navigation (graph, controller)
XML
Layout
(View,
ViewGroup)
Activity
Fragmentmanager
XML
Layout
Fragment
X
XML
Layout
Fragment
intent
XML
Layout
Activity
Observer
View Model (LiveData)
Observable
Model
(Data
Layer)
Data
Access
(CRUD)
Repository
Local Database
(DAO)(Sqlite, Room)
Local
Files
Network data
(RESTAPI, firestore)
Content
Provider
Data

ViewModel and LiveData

- Subclass the ViewModel class
- Define containing Livedata
  - Mutable - can change value (private)
  - Immutable - only get value (public)
  - SetValue() (value assignment) triggers notification to observers.
- Create ViewModel using ViewModelProvider in UI controllers (Activity or Fragment)
  - Create a ViewModelProvider with the StoreOwner (scope) specified.
  - Get a specific view model
- Create observers for livedata defined in ViewModel in ViewModel in UI controllers
  - Define how to load livedata value into UI components in the onChanged() method of the Observer interface.

Fragment Communication using ViewModel

- The ViewModel remains in memory until the ViewModelStoreOwner to which it's scoped goes away permanently. Once a ViewModel object is first instantiated, the subsequent calls to retrieve the ViewModel using the same scope always returns the same existing ViewModel along with the existing data until the ViewModelStoreOwnerStoreowner’s lifecycle has permanently ended.
- Fragment communication through shared ViewModel
  - Use the same scope (activity, parent fragments, backstackentry)
  - Use the same ViewModel class.

Adaptive to Different Screen Size and Orientation

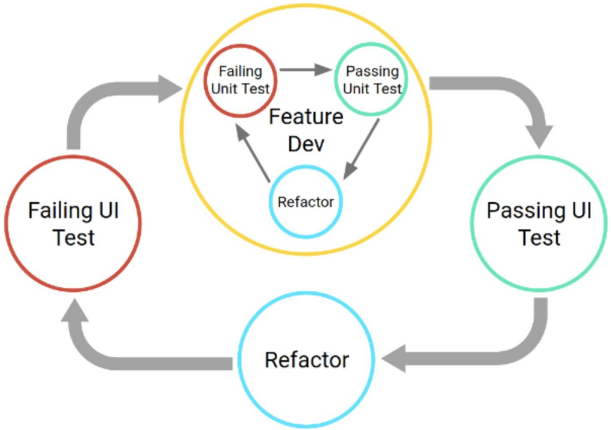
- Use view dimensions that allow the layout to resize (constraint layout, weight in Linear layout, avoid hard code size, match_parent, wrap_ content for view size)
- Create alternative UI layouts according to the screen configuration (create resource directory folders with different qualifier name: res/layout, res/layout-sw600dp, res/layout-land/, res/layout-w600dp, res/layout-large/ ...)
- Provide bitmaps that can stretch with the views (nine patch)
- https://developer.android.com/training/multiscreen/screensizes

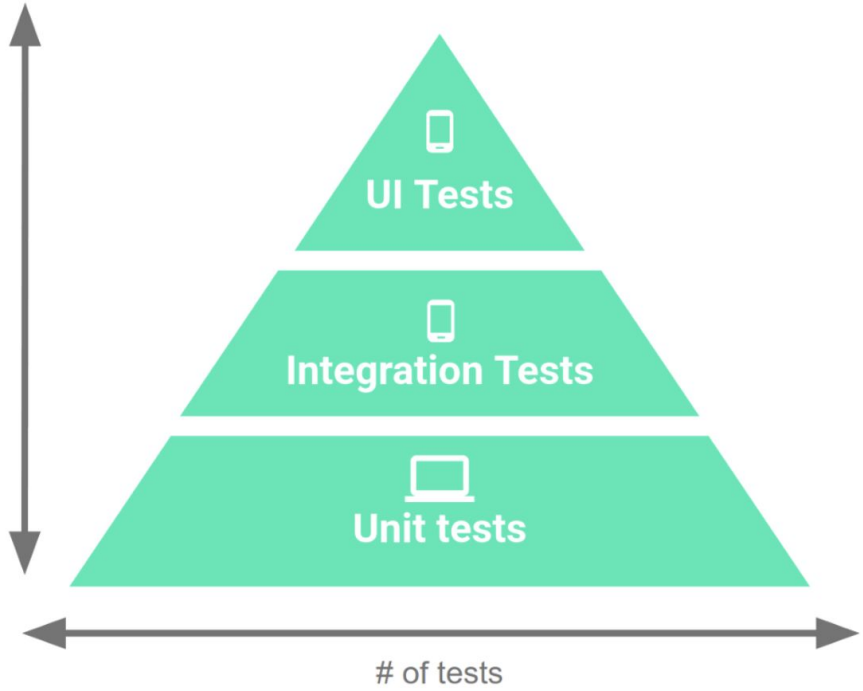
SlidingPaneLayout

- SlidingPaneLayout provides a horizontal, multi-pane layout for use at the top level of a UI. A left (or start) pane is treated as a content list or browser, subordinate to a primary detail view for displaying content.
- SlidingPaneLayout should be thought of only as a way to allow a two-pane layout normally used on larger screens to adapt to smaller screens in a natural way.
- Child views overlap if their combined width exceeds the available width in the SlidingPaneLayout. Each of child views is expand out to fill the available width in the SlidingPaneLayout. When this occurs, the user may slide the topmost view out of the way by dragging it, and dragging back it from the very edge.

Testing

- Small tests (unit)
- Medium tests (integration)
- Large tests (end-to-end UI)





https://developer.android.com/training/testing/fundamentals

Running Tests

- Run on the local computer, in the *test* folder.
  - Faster
  - Tests that don’t involve any Android resources
  - Tests that involve Android resources
    - Simulated devices (such as Robolectric)
- Run on the real or virtual mobile devices, in the *androidTest* folder
  - Slower
  - Tests that involve Android resources
  - With higher fidelity

UI Tests

- Use Espresso to write concise, beautiful, and reliable Android UI tests.
- Espresso tests state expectations, interactions, and assertions clearly.
- The AndroidJUnitRunner class defines an instrumentation-based JUnit test runner that lets you run JUnit 3- or JUnit 4-style test classes on Android devices. The test runner facilitates loading your test package and the app under test onto a device or emulator, running your tests, and reporting the results.
- ActivityScenarioRule launches a given activity before the test starts and closes after the test.

Questions?