

CS683 Mobile Application Development

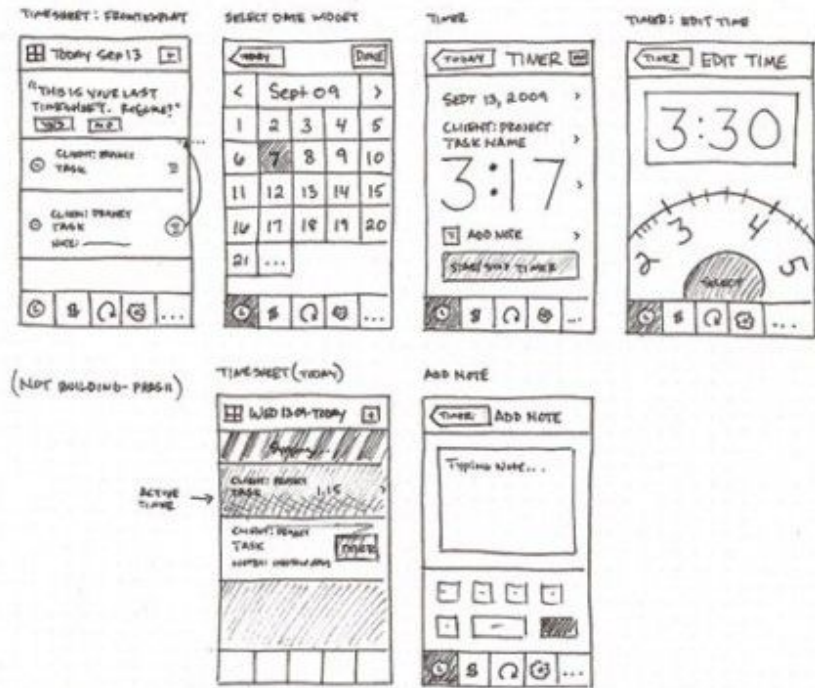
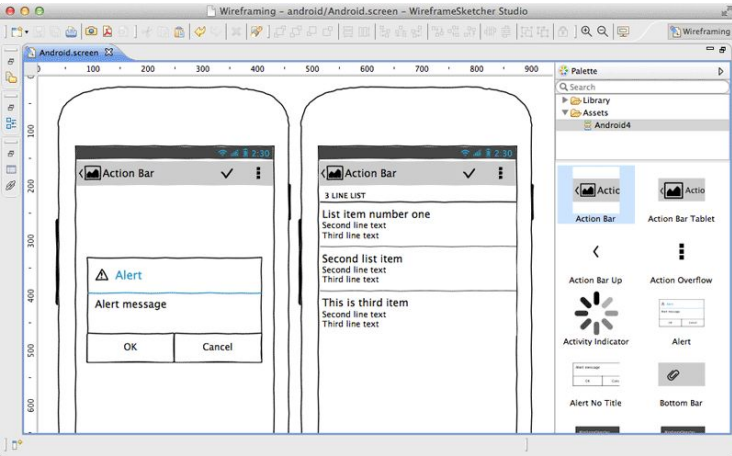
User Interfaces

Yuting Zhang
BU METCS

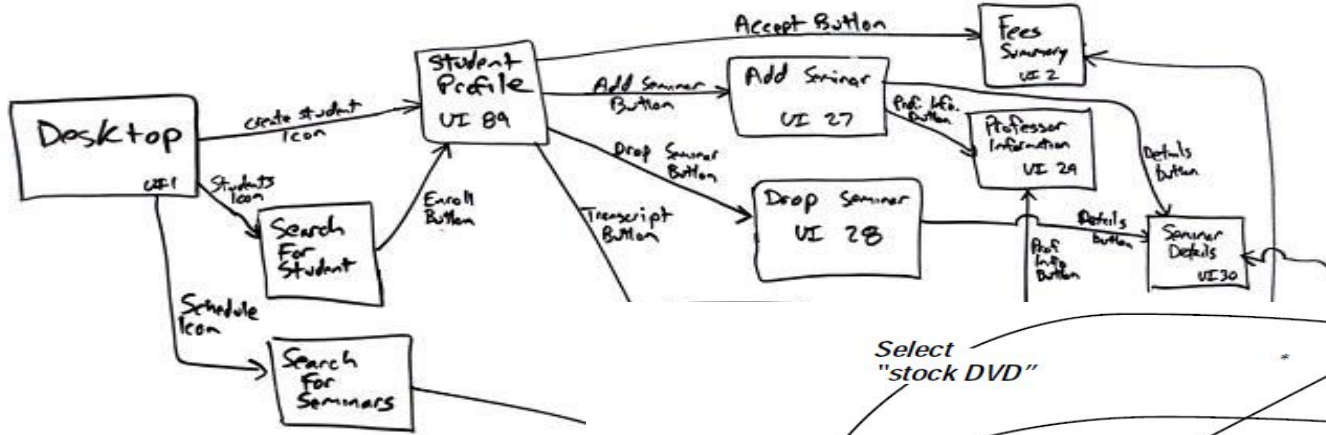
User Interface Design

- UI: a collection of visual objects arranged on the screen that users can interact with.
- In both requirement analysis and design phase
- Lo-Fi UI Mockups (Lo-fi wireframe)
 - Pen, paper (post-it-note), balsamiq
 - Just mock, convey the idea not detail (color, font, pixels)
 - Fast, cheap, brainstorming, early feedback, maintainability, modularity.
- UI navigation: show how UI changes based on user actions
 - Choose the right navigation path
 - Goal: ease of use

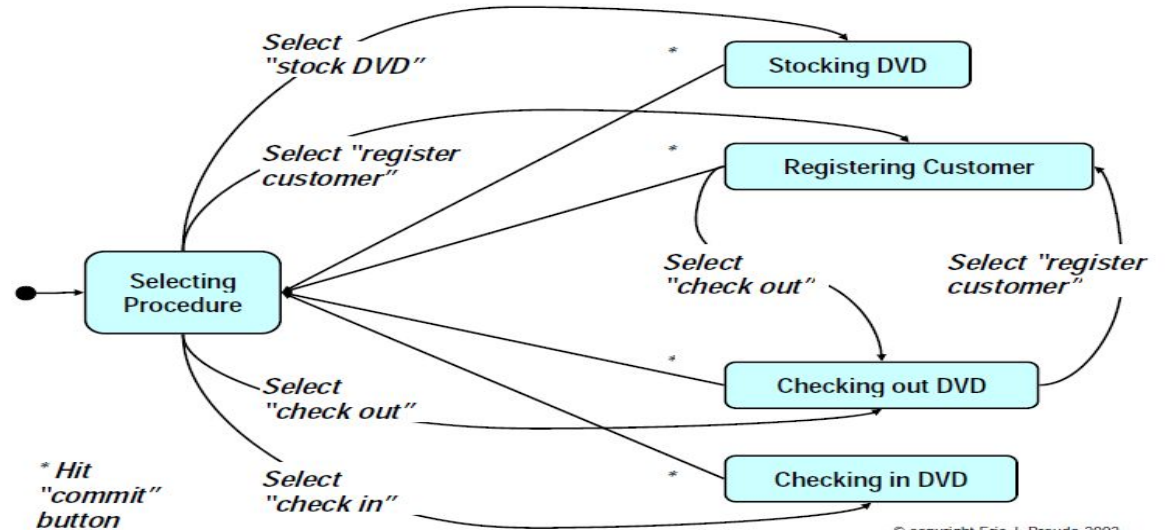
UI Mockups



UI Navigation



UI Flow Diagram



State Transition Diagram

User Interface

- Design challenges
 - Adapt to different screen sizes and orientations
 - Performance (transition smooth, no glitch), data leakage
- How to design/implement UI
 - The look: hierarchy layout + components (widgets)
 - The events and actions: response to users' action
 - UI navigation: move from one screen to another
 - Communication: data pass from one screen to another
- How to maximize its reusability and flexibility
 - Activity vs. fragment
- Separation of Data and its Presentation (MVC, MVVM ...)

UI Elements

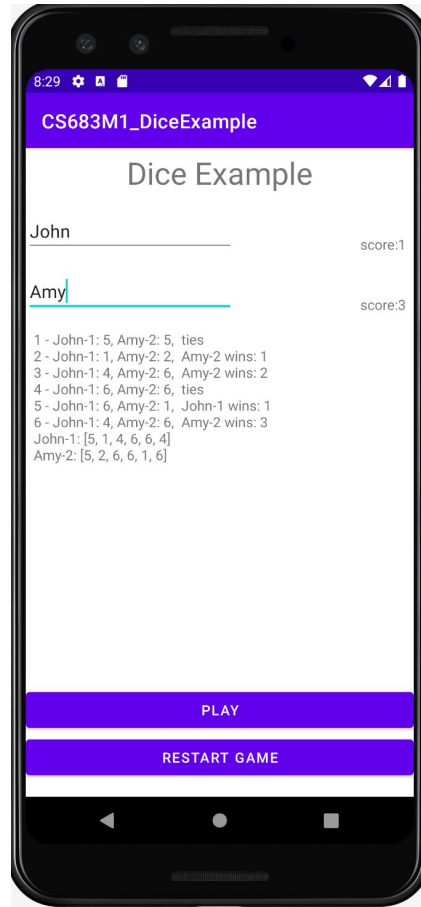
Look:

Labels (Text)

Input fields

Buttons

They are organized together in certain ways



User interactions:

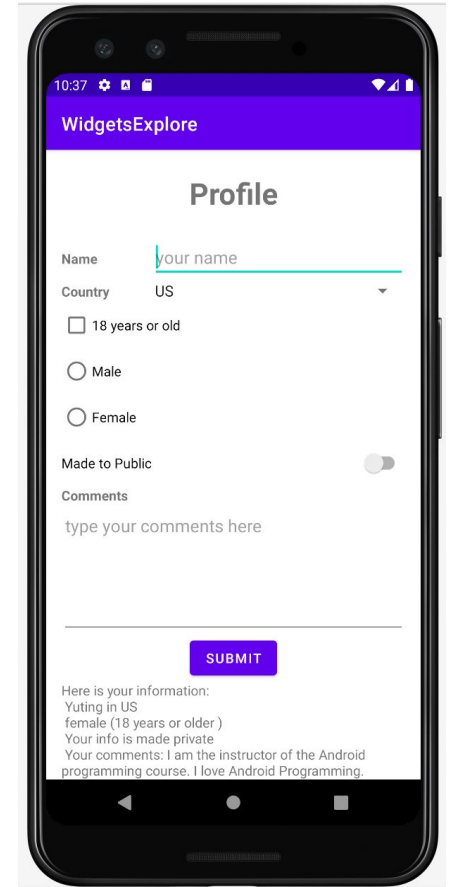
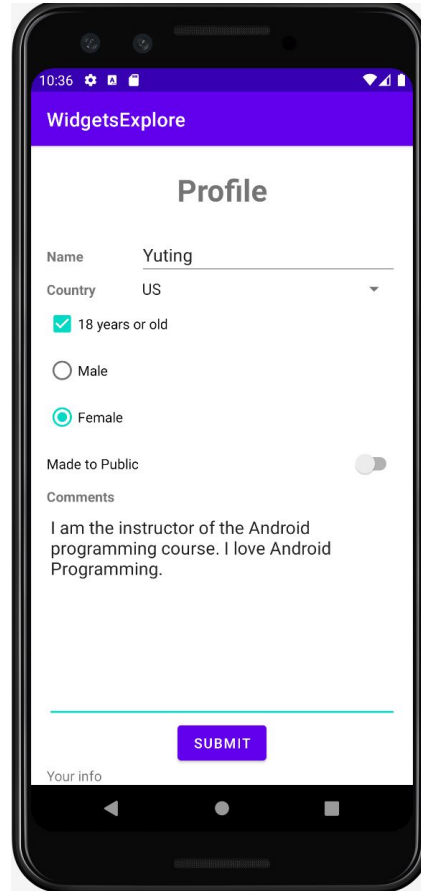
Input fields: users can enter data. Data can be displayed in other text fields

Buttons: users can click on the button, and generate data in other text fields.

Example: WidgetsExplore

See Code:

https://github.com/CS683/METCS683/blob/main/CodeExamples/CS683M2_WidgetsExploreinkt/



XML Layout Files

- Consists of a hierarchy of UI components objects
- Define the initial attributes of each component
 - id
 - layout_width
 - layout_height
 - Text
 - ...
- <https://developer.android.com/guide/topics/ui/declaring-layout.html>
- Can use AS layout editor

```
<?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" >  
  
    <TextView android:id="@+id/text"  
  
        android:layout_width="wrap_content"  
  
        android:layout_height="wrap_content"  
  
        android:text="Hello, I am a TextView" />  
  
    ...  
  
</LinearLayout>
```


Pixels and Spacing

- dp (Density-independent Pixels)
 - An abstract unit that is based on the physical density of the screen - relative to a **160 dpi (dots per inch) (1 dp is about 1/160 inch)**
 - When running on a higher density screen, the number of pixels used to draw 1dp is scaled up by a factor appropriate for the screen's dpi - likewise, when on a lower density screen, the number of pixels used for 1dp is scaled down
 - The ratio of dp-to-pixel will change with the screen density, but not necessarily in direct proportion
 - Using dp units (instead of px units) is a simple solution to making the view dimensions in your layout resize properly for different screen densities. In other words, it provides consistency for the real-world sizes of your UI elements across different devices.

Pixels and Spacing

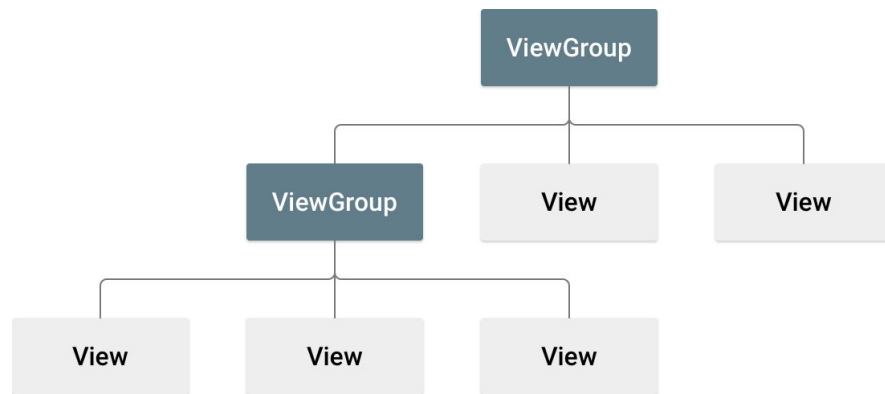
- SP(Scale-independent Pixels)
 - This is like the dp unit, but it is also scaled by the user's font size preference.
 - It is recommended to use this unit when specifying font sizes, so they will be adjusted for both the screen density and the user's preference
- Pt (Points)
 - 1/72 of an inch based on the physical size of the screen
- Px (Pixels)
 - Not recommended because the actual representation can vary across devices; each devices may have a different number of dpi (pixels per inch) and may have more or fewer total pixels available on the screen

Android UI Classes

- Activity: A UI entry component
- **View:** A View is an object that draws something on the screen that the user can interact with.
 - A rectangular area of the display
 - For examples: Textview, imageview, button, listview, menuview, progressbar ... (all kinds of widgets)
 - Each view has a number of attributes that defines its internal state. These attributes can be set to change its state.
 - Common attributes: id, layout_width, layout_height,...
- <https://developer.android.com/guide/topics/ui/overview.html>

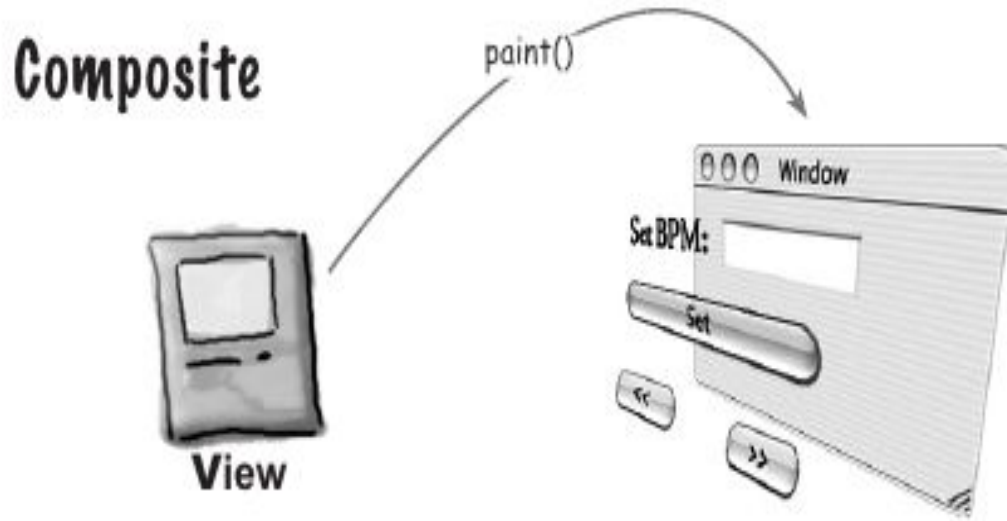
Android UI Classes

- Activity: UI entry component
- View: A View is an object that draws something on the screen that the user can interact with.
- ViewGroup: A ViewGroup is an object that holds other View (and ViewGroup) objects in order to define the layout of the interface.
 - Layouts: CoordinatorLayout, FrameLayout, GridLayout, **ConstraintLayout** ...
 - View containers: AdapterView
 - A special view
- <https://developer.android.com/guide/topics/ui/overview.html>



Composite Pattern

- Models tree structures that represent part-whole hierarchies with arbitrary depth and width.
- The Composite Pattern treats individual objects and compositions of these objects uniformly

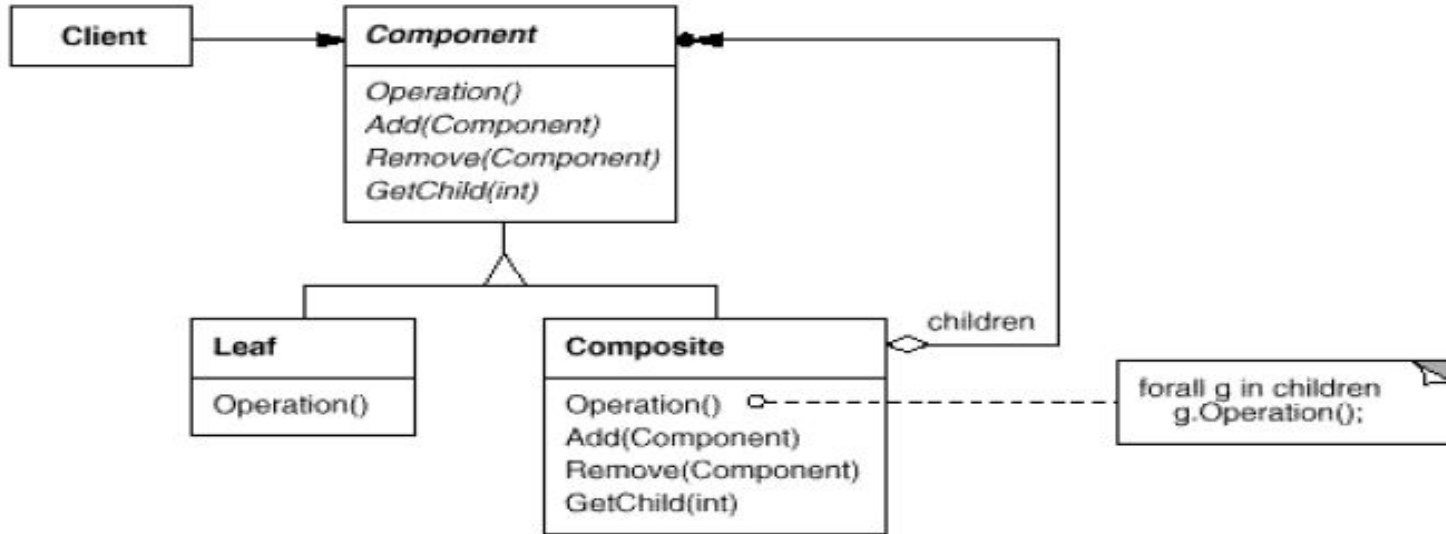


The view is a composite of GUI components (labels, buttons, text entry, etc.). The top level component contains other components, which contain other components and so on until you get to the leaf nodes.

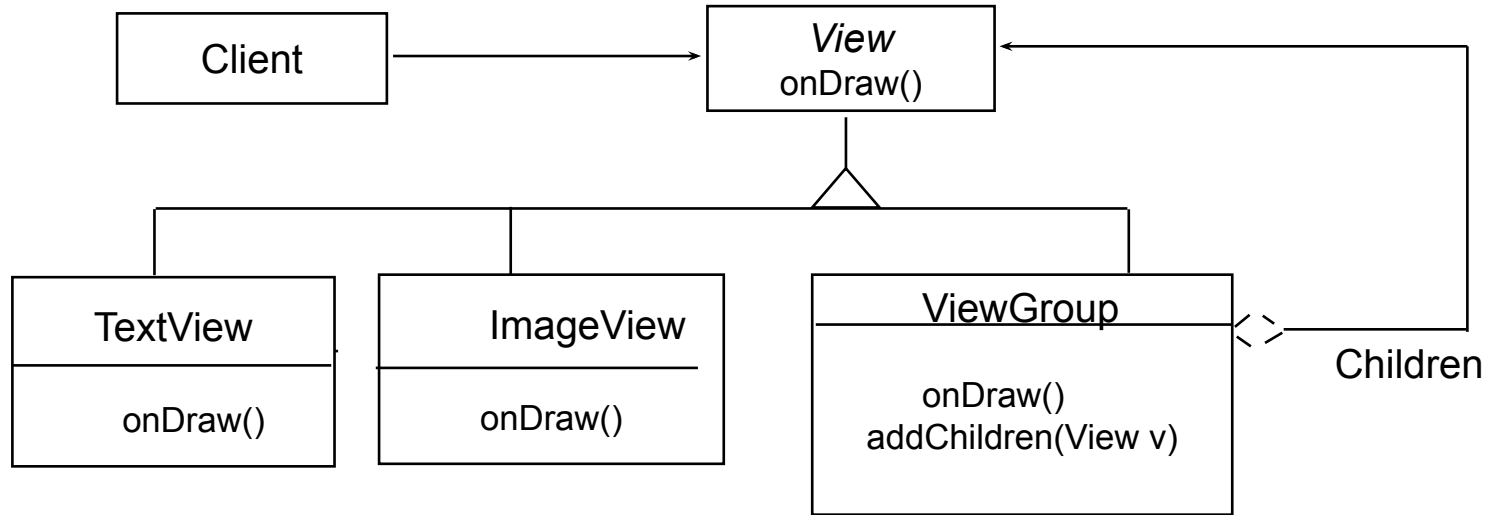
“Head First Design Pattern”

Composite Pattern

- Displayed in the component Tree” Windows in AS.



Composite Pattern Used by the View Class



Simple Widgets

- Subclasses of the View Class
- Android provides a lot of widgets. Each widget inherits some properties from the parent class as well as has its own properties.
- Text I/O (TextView, EditText)
 - PlainText, Password, PersonName, Email, etc
- Button, Toggle Button, Switch, , CheckBox, RadioButton
- ProgressBar, SeekBar, RatingBar, etc
- Spinner

Ab TextView

 Button

 ToggleButton

 CheckBox


 **RadioButton**


A  CheckedTextView

 Spinner


 ProgressBar

 ProgressBar (Horizontal)


 SeekBar

 SeekBar (Discrete)

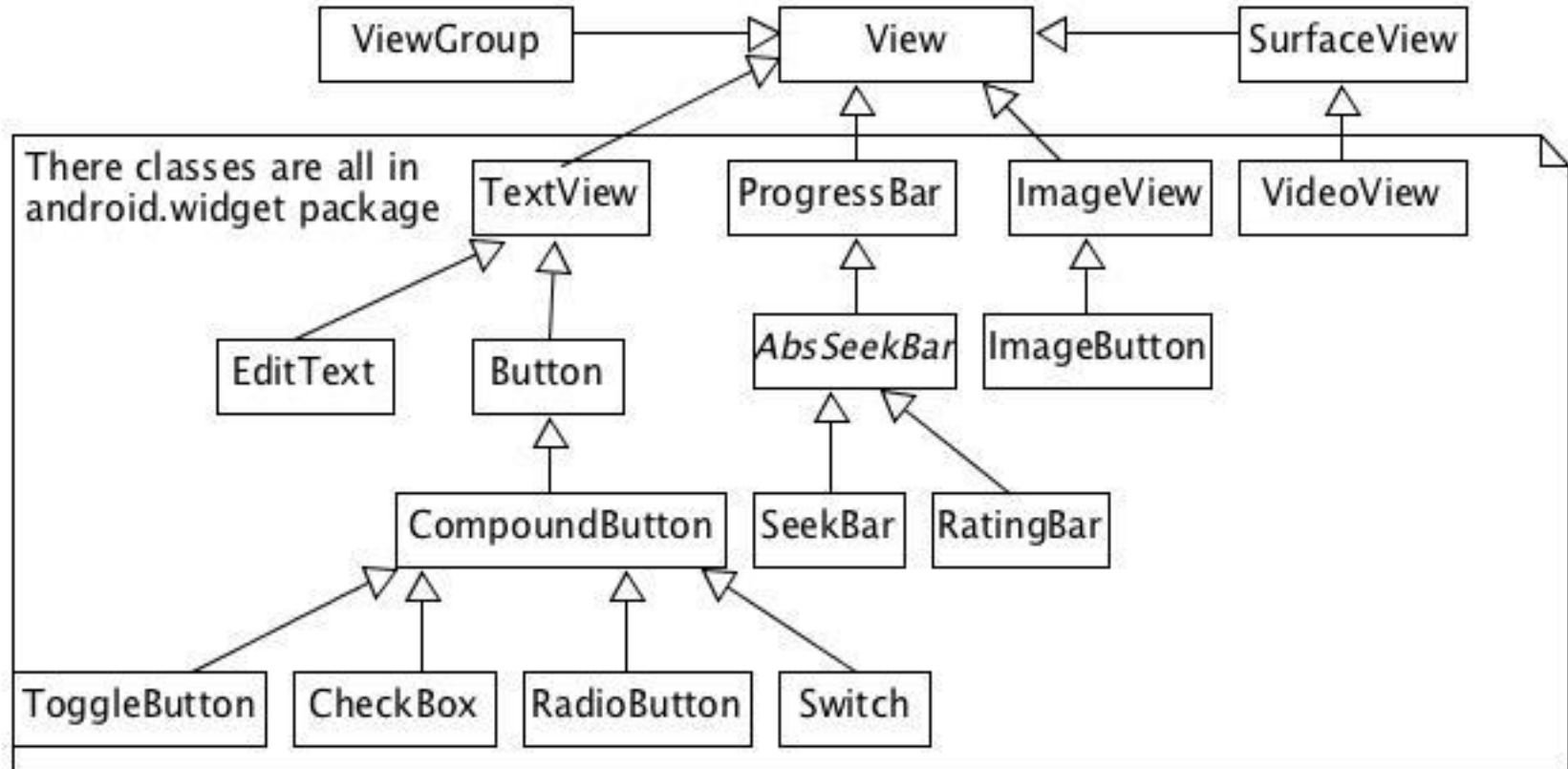
 QuickContactBadge

 RatingBar

 Switch










 Space

Class Diagram for Basic Widgets



Containers

- Subclasses of ViewGroup
- Act as containers for other view Widgets
- RadioGroup: container of multiple RadioButtons

-  RadioGroup
-  ListView
-  GridView
-  ExpandableListView
-  ScrollView
-  HorizontalScrollView
-  TabHost
-  WebView
-  SearchView

Layouts

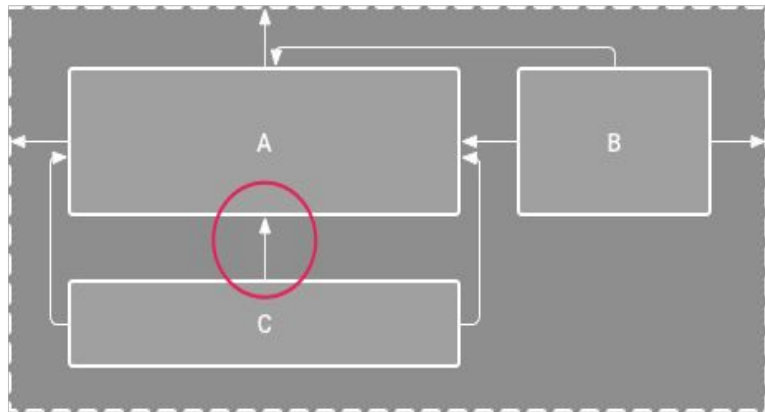
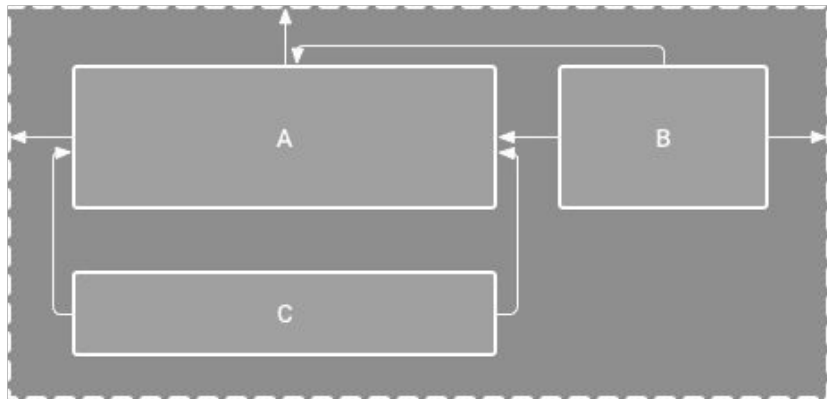
- A layout is subclassed from ViewGroup. It has children components in it and their relationships.
- LinearLayout
 - Align all children in a single direction, vertically or horizontally.
- TableLayout
 - Display child View elements in rows and columns.
- **ConstraintsLayout**
- FrameLayout
- CoordinatorLayout

ConstraintLayout

- Creates large and complex layouts with a flat view hierarchy (no nesting)
- Similar to RelativeLayout in that all views are laid out according to relationships between sibling views and the parent layout. Replace RelativeLayout in the template.
- More flexible and easier to use with Android Studio
- To define a view's position in ConstraintLayout, you must add at least one horizontal and one vertical constraint for the view.
- Each constraint represents a connection or alignment to another view, the parent layout, or an invisible guideline
- <https://developer.android.com/training/constraint-layout>

ConstraintLayout

- A's left and top are constrained by its parent
- B's left are constrained by A, B's right by the parent
- B's top is aligned with A
- C's left and right are aligned with A
- C's top is constrained by A.



Good Tutorials

- <https://www.youtube.com/watch?v=z53Ed0ddxgM>
- https://www.tutorialspoint.com/android/android_user_interface_layouts.htm

FrameLayout

- Allocate an area on the screen to display a single item.
- Generally, FrameLayout should be used to hold a single child view. If multiple child views are added they will, by default, appear on top of each other positioned in the top left hand corner of the layout area.
- Alternate positioning of individual child views can be achieved by setting gravity values on each child.

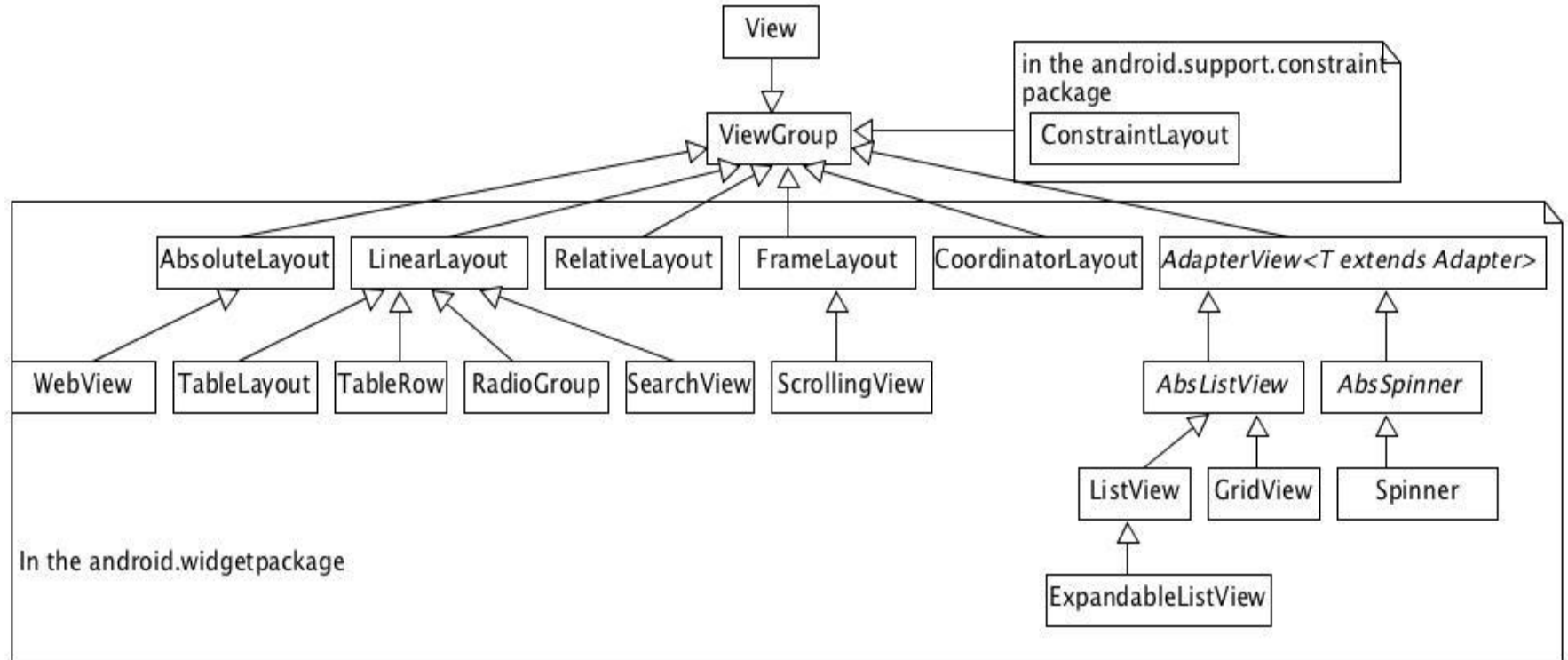


https://www.tutorialspoint.com/android/android_frame_layout.htm

CoordinatorLayout

- Introduced as part of the Android Design Support Library with Android 5.0
- Designed specifically for coordinating the appearance and behavior of the app bar across the top of an application screen with other view elements
- When creating a new activity using the Blank Activity template, the parent view in the main layout will be implemented using a CoordinatorLayout instance.
- CoordinatorLayout is a super-powered FrameLayout.
- Used as a top-level application decor or chrome layout
- Used as a container for a specific interaction with one or more child views

Class Diagram for ViewGroup



Layout Files

- A layout XML file (in res/layout/) defines the visual structure for a user interface. (Define the initial attributes of all view components)
- Two ways to create
 - Declare UI elements in XML. (Easy, visualized, better separation and customization)
 - Can use different xml layout files for different screen sizes, different orientations.
 - Instantiate layout elements at runtime. Create View and ViewGroup objects in Java/Kotlin code. (dynamic change)
- What are pros and cons of using the XML layout file?
- <https://developer.android.com/guide/topics/ui/declaring-layout.html>

Use XML Layout Files

- First, declare an XML file using the text or design mode
- Load the XML file in Activity.

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_main)  
}
```

- Each individual view object can be referenced by its ID in Activity.
Manipulate each view object individually as needed through its attributes.

```
val myButton = findViewById<Button>(R.id.my_button);
```

Activity vs. View

- Each activity is usually mapped to one screen.
- Each screen is usually defined by a layout file that describe a hierarchy of view objects.
- The View Class defines what the UI looks like, and can listen to various UI events.
- The user interaction code is usually written in the corresponding Activity class. The View object references are acquired based on their id.

UI Events Handling

- UI events are user's interactions, also called input events.
- To handle the event, Views must register the appropriate event listener and implement the corresponding callback.
- An event listener is an interface in the View class that contains a single callback method.
- Android captures events from the specific View object that the user interacts with.
- Android puts events into an event queue (usually FIFO).
 - Passed to the view where the event took place
 - Includes the nature of the event (coordinates, type, etc.)and
- Then calls the implemented callback function.

EventListeners and Callbacks

- `onClickListener (onClick())`: detect click events (touches and release)
- `onLongClickListener (onLongClick())`: detect when the user maintains the touch for an extended period
- `onTouchListener (onTouch())`: detect any form of contact with the touch screen including individual or multiple touches and gesture motions
- `onCreateContextMenuListener (onCreateContextMenu())`: Listen for the creation of a context menu as the result of a long click
- `onFocusChangeListener (onFocusChange())`: detect when focus moves away from the current view (track-ball or navigation key)
- `onKeyListener (onKey())`: detect when a key on a device is pressed while a view has focus

Register Listeners and Implement Callbacks

- Register an eventListener via set....Listener(), and implement callback.

```
open fun setOnClickListener(l: View.OnClickListener?): Unit
```

```
findViewById<Button>(R.id.buttonId_submit)?.setOnClickListener {  
    findViewById<TextView>(R.id.textView).text = "button clicked"  
}
```

Use View Event onClick Attribute

- Assign the callback method in the view event onClick attribute directly in XML layout file

```
<Button  
    android:id="@+id/button1"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:onClick="buttonClick"  
        android:text="Click me" />
```


Event Handlers

- **onClick(v: View)**
- **onLongClick(v: View): Boolean** returns a Boolean to indicate if the event was consumed
 - If returns true, the event is discarded by the Android framework
 - If returns false, the event is still active and is passed along to the next matching event listener
- **onKey(v:View, keyCode: Int, event: KeyEvent):Boolean**
- **onTouch(v: View , m: MotionEvent): Boolean**
 - The MotionEvent object is the key to obtain the information about the event such as the location of the touch and action type (e.g. ACTION_DOWN. ACTION_UP, ACTION_MOVE)
 - Can have multiple actions that follow each other

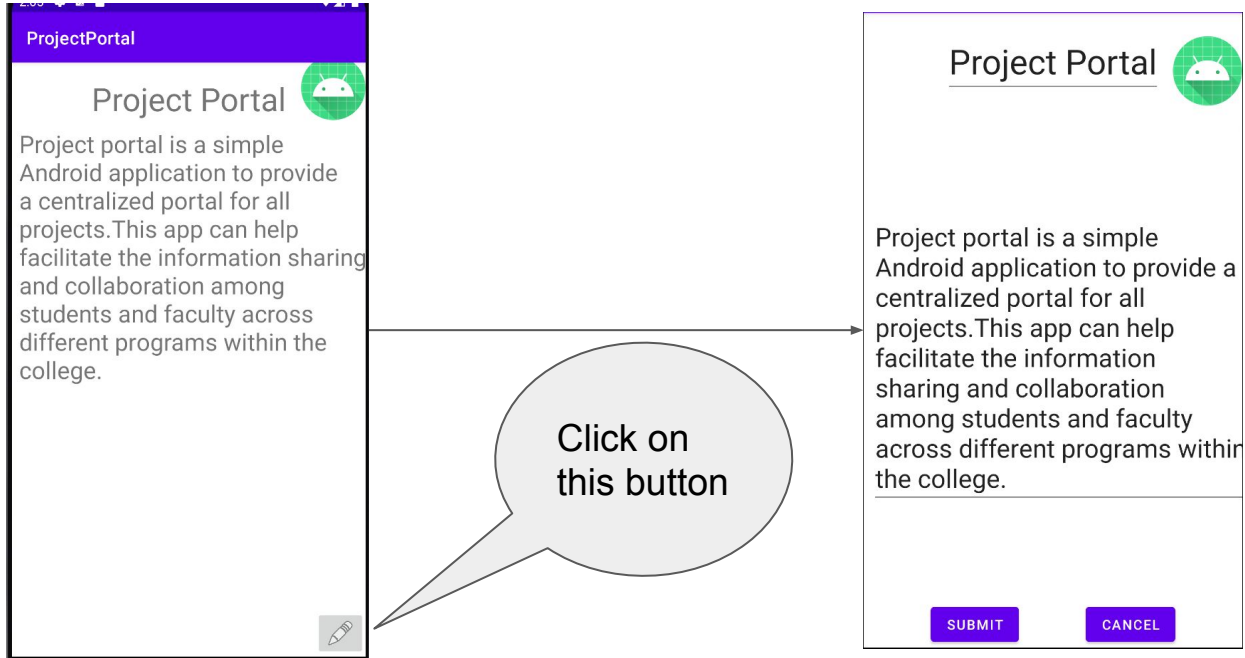
Other Event Handlers

- `onKeyDown(int, KeyEvent)`: Called when a new key event occurs
- `onKeyUp(int, KeyEvent)`: Called when a key up event occurs
- `onTrackballEvent(MotionEvent)`: Called when a trackball motion event occurs
- `onTouchEvent(MotionEvent)`: Called when a touch screen motion event occurs
- `onFocusChanged(boolean, int, Rect)`: Called when the view gains or loses focus
- You can build custom view component and define these callbacks.

UI Navigation

- Usually an application can have multiple screens
- The user can navigate through those screens
- We can implement each screen as an activity
- Use inter-component communication to navigate.

Example: Project Portal



Intent for Communication

- An Intent is a messaging object you can use to request an action from another app component.
- Traditionally messaging in OOP when obj1 sends an message to obj2, then in obj1 will call obj2.method(parameters).
 - This is more static. Need specify obj2 and the method name in the program.
- An Intent message encapsulates receiving components, action(method) and data(parameters) together.
-

`Intent()`

Create an empty intent.

`Intent(o: Intent!)`

Copy constructor.

`Intent(action: String!)`

Create an intent with a given action.

`Intent(action: String!, uri: Uri!)`

Create an intent with a given action and for a given data url.

`Intent(packageContext: Context!, cls: Class<*>!)`

Create an intent for a specific component.

`Intent(action: String!, uri: Uri!, packageContext: Context!, cls: Class<*>!)`

Create an intent for a specific component with a specified action and data.

Inter-Component Communication

- Send Intent from one Android component to another (Activity, Service, Broadcast Receiver)
- Defined in the Context Class:
 - **Start an activity:** `startActivity(intent)`
 - **Start a service:** `startService(intent)`, `bindService(intent)`
 - **Send a broadcast intent:** `sendBroadcast(intent)`
- An Activity is also a Context object.

In the main Activity:

```
editProj.setOnClickListener {  
    startActivity(Intent(this, EditProjectActivity::class.java))  
}
```

Reusability and Flexibility

- Activities are independent component. If multiple screens have similar parts, and they cannot be reused.
- Once a layout file is inflated in an activity, the appearance is harder to change.
- Fragments are introduced in Android 3.0 (Use androidx fragment for backward compatibility).
 - Enable reuse, support more dynamic and flexible UI designs.
 - Able to modify the activity's appearance at runtime and preserve those changes in a back stack that's managed by the activity

Why Fragments?

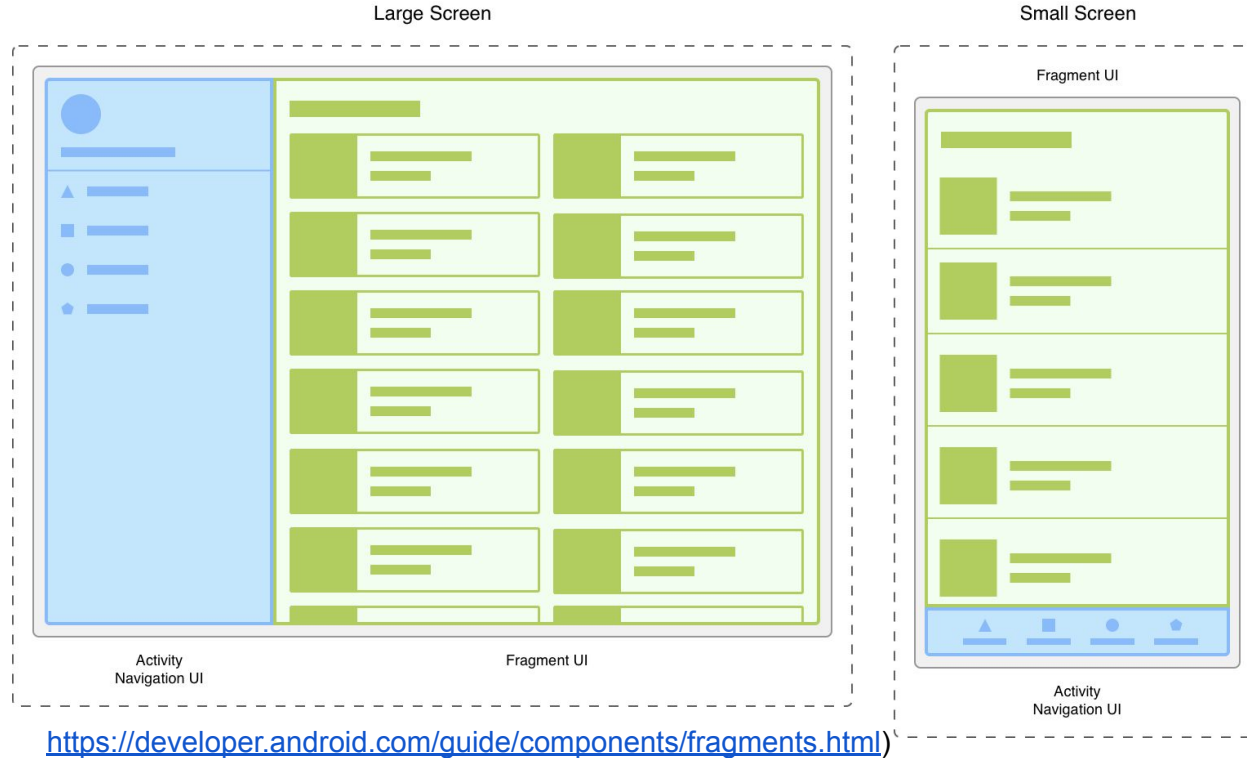


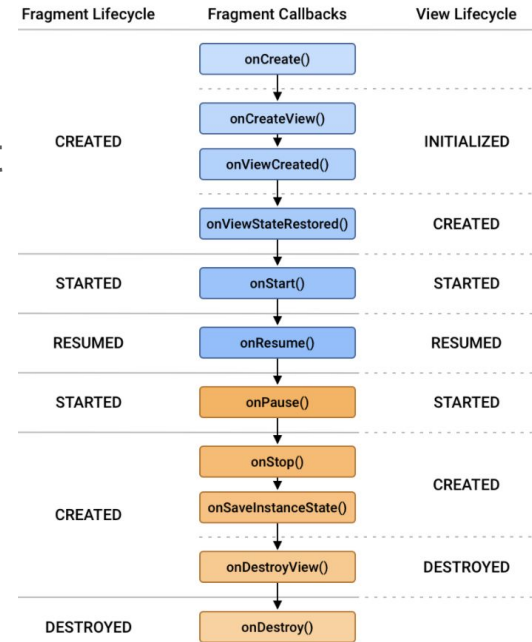
Figure 2.1. Two versions of the same screen on different screen sizes. On the left, a large screen contains a navigation drawer that is controlled by the activity and a grid list that is controlled by the fragment. On the right, a small screen contains a bottom navigation bar that is controlled by the activity and a linear list that is controlled by the fragment.

Fragments

- Self-contained, modular section of an application's user interface and corresponding behavior that can be embedded within an activity
 - Sort of like a “sub activity” (But a fragment is NOT a type of activity). Usually associated with a XML and a Java/Kotlin file.
 - Multiple fragments can be combined in a single activity to build a multi-pane UI.
 - has its own lifecycle, receives its own input events,
 - Can be added or removed to create a dynamically changing user interface at runtime.
 - Can be reused in multiple activities.
 - Cannot be instantiated as standalone application elements, only used as part of an activity
 - <https://developer.android.com/guide/components/fragments.html>

Fragment Lifecycle

- Directly affected by the host activity lifecycle
 - When the activity is paused, so are all fragments in it
 - When the activity is destroyed, so are all fragments in it
- When an activity is running, each fragment can be in different state.
 - A fragment can be added
 - A fragment can be removed
 - A fragment can be added to the back stack
 - A fragment can return to the layout from the back stack. (e.g. navigate backwards by pressing the Back button)



Create A Fragment

- Usually consist of two components (very similar to activity)
 - Create a layout for a fragment
 - Create a subclass of Fragment (or an existing subclass of it).
 - Load the layout file using inflater in onCreateView() callback.
 - Implement other callback methods such as onCreate(), onStart(), onPause(), and onStop() as needed

```
class DetailFragment : Fragment() {  
    override fun onCreateView(  
        inflater: LayoutInflater, container: ViewGroup?,  
        savedInstanceState: Bundle?): View {  
        return inflater.inflate(R.layout.fragment_detail, container, false)  
    }  
}  
  
class DetailFragment: Fragment(R.layout.fragment_detail)
```

Where the fragment
will be placed in

Add A Fragment via the Activity's XML Layout

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.fragment.app.FragmentContainerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:name="edu.bu.projectportal.DetailFragment"
    android:layout width="match parent"
    android:layout height="match parent"
    tools:context=".MainActivity" />
```

Add A Fragment Programmatically

- Use Fragment Manager in the activity
 - Use `supportFragmentManager`
 - Use `Fragment-ktx` for Kotlin

```
implementation "androidx.fragment:fragment-ktx:1.2.5"
```

```
override fun editProj () {  
    supportFragmentManager.commit {  
        replace<EditFragment>(R.id.container)  
    }  
}
```

<https://developer.android.com/guide/fragments/fragmentmanager>

Handling Events inside Fragment

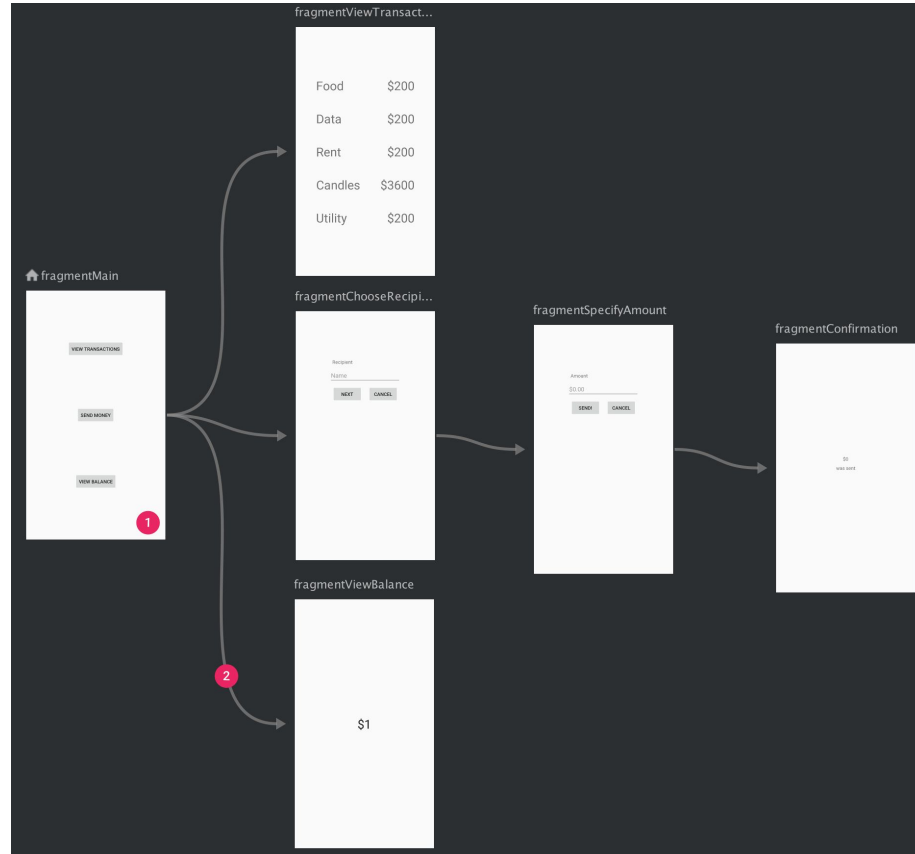
- The view components within a fragment can listen to events just like those in a regular activity.
- Whether the fragment or the activity should handle such events depends on how the event handler is declared.
- The general rule for events generated by a view in a fragment is that if the event listener was declared in the fragment class using the event listener and callback method approach, then the event will be handled first by the fragment
- But if the `android:onClick` resource is used, the event will be passed directly to the activity containing the fragment.

Navigation

- We can use the JetPack Navigation Component to simplify the communication among fragments/activities (
- Create a navigation graph as a resource file (res/navigation/nav_graph.xml)
 - Define start destination (fixed): the first and last screen
 - Define the screen navigation and actions (add destinations)
 - Back stack: the top one is the current screen, and the bottom one is the start destination.

<https://developer.android.com/guide/navigation>

Navigation



Navigation

- We can use the JetPack Navigation Component to simplify the communication among fragments
- Create a navigation graph as a resource file
- Add a NavHost (the container for destinations) to an activity
 - Derive from NavHost.
 - Default NavHost implementation: NavHostFragment, handles swapping fragment destinations.
- Navigate to a destination using NavController
 - `view.findNavController().navigate(R.id.actionid)`

Navigation

- Move from one screen to another, and move back
 - Back stack
- Between Activities
 - Through Intents
- Between Fragments
 - Indirect through containing activities.
 - Using Fragment Manager
- Using the JetPack Navigation component:
 - Navigate correctly, highlight right button, handle back stack
 - https://developer.android.com/guide/navigation?gclid=CjwKCAjwmK6lBhBqEiwAocMc8uu-eUZxNtMaADou6Uxen6zAigiEhQVIWPPQ2EdOgZ-pTJ5DLcZhPhoCCSEQAvD_BwE&gclidsrc=aw.ds
 - <https://www.youtube.com/watch?v=IEO2X5OU3MY>
 - <https://github.com/mitchtabian/Navigation-Components-Example>

Questions?