2.3 User Interfaces

Accessibility and Navigation

Objectives

- Make apps more accessible
- Build a dynamic and multi-pane UI using Fragment
- Create Dialog
- Create Navigation Drawer
- Create Tabs

Accessibility

- Accessibility: regardless of ability, users are able to navigate, understand, and use an app successfully

- Consideration

Navigation

Readability

Guidance and Feedback

https://developer.android.com/guide/topics/ui/accessibility/apps

Accessibility - Navigation

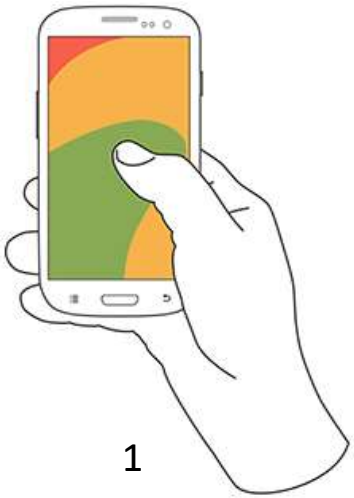
1. Support screen readers by:
   - label UI elements
   - provide pane titles (for API level 28)
   - group related content

2. Make touch targets at least 48 x 48 pixel.
   Space between elements of your mobile design should be at least 8dp.

3. Support voice and gesture navigation

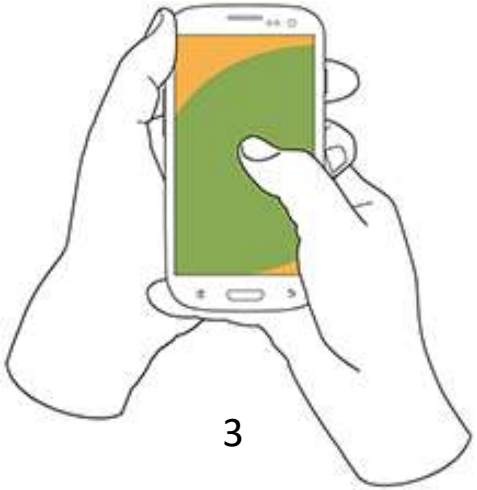
Question?

How users hold their phones?

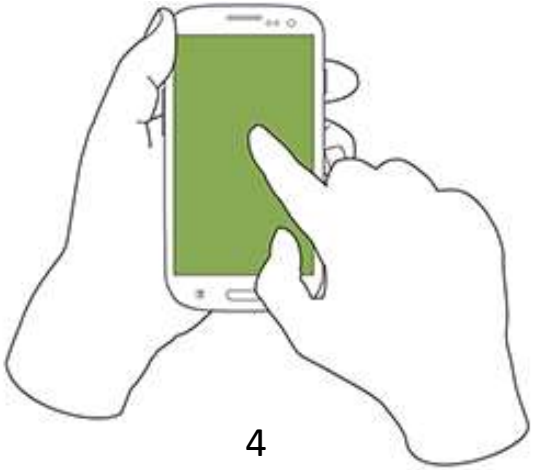


1

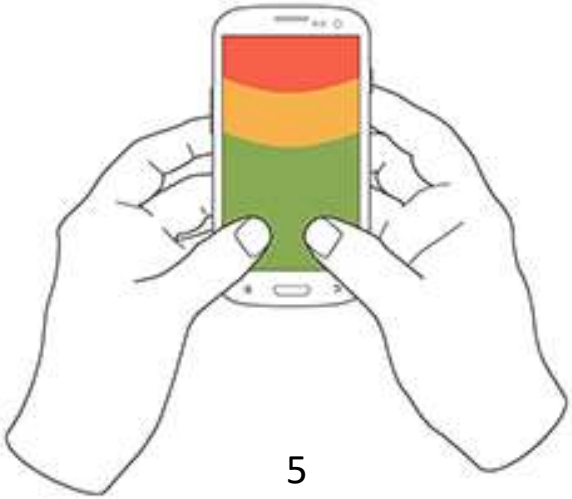
2



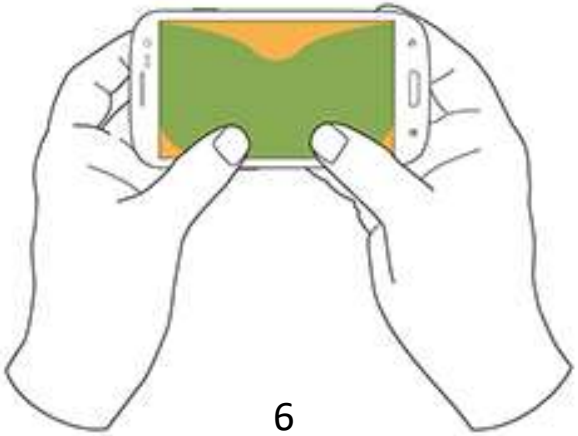
3



4

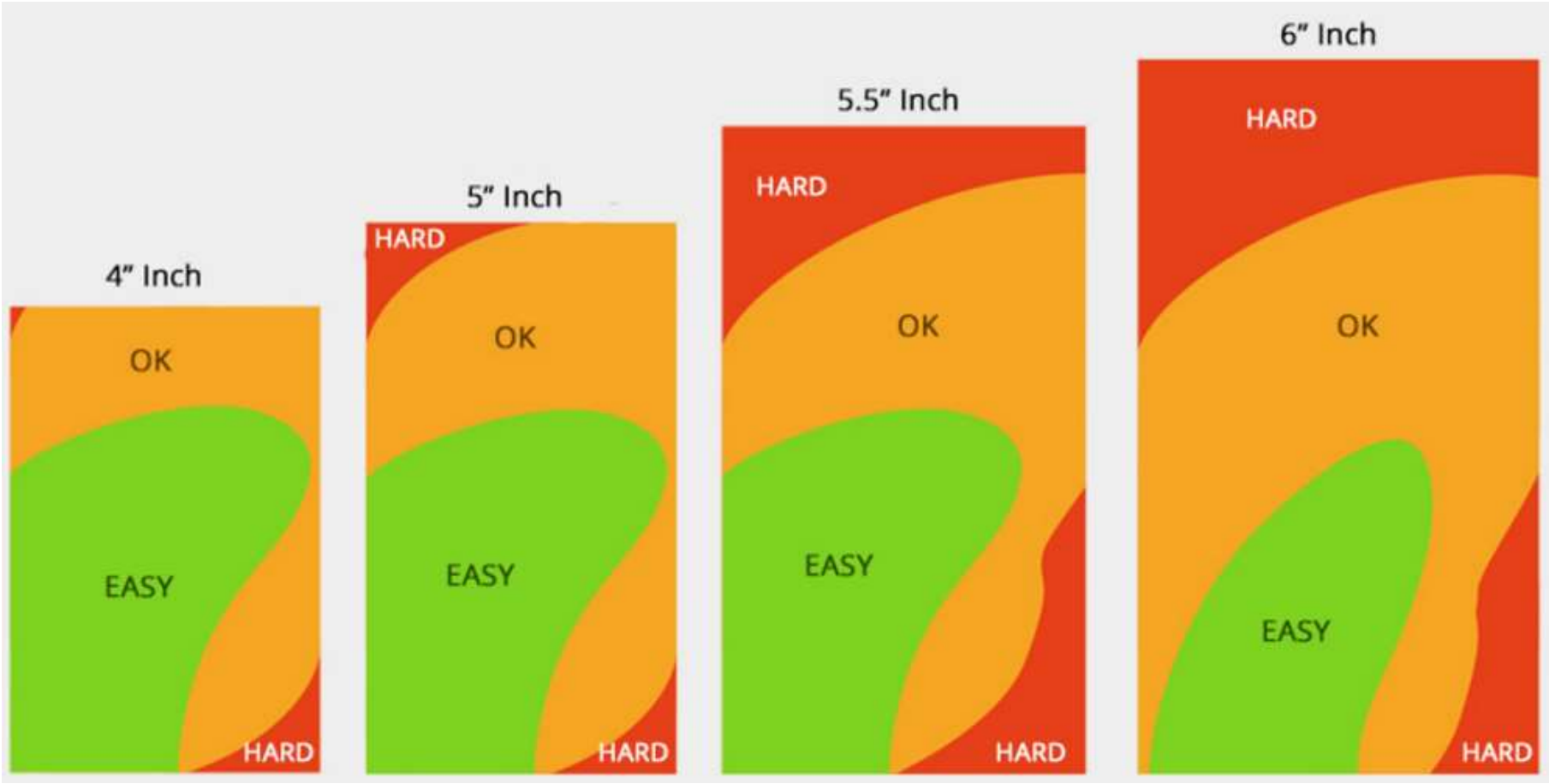


5



6

Touch Zone



https://www.probeseven.com/blog/mobile-uiux-design-considerations-outdoors-visually-challenged/

Gesture Navigation

- Android 10 (API 29) supports fully gestured-based navigation

- Ensure your apps:
  1. Extend content from edge to edge
  2. Handle conflicting gestures

https://developer.android.com/guide/navigation/gesturenav

Gesture Navigation

- Extend content from edge to edge
  - Set transparent system bars


```
<!-- values-29/themes.xml: -->

<style name="AppTheme" parent="...">
    <item
name="android:navigationBarColor">@android:color/transparent</item>

    <!-- Optional, but recommended for full edge-to-edge rendering -->
    <item name="android:statusBarColor">@android:color/transparent</item>
</style>
```

Accessibility - Navigation

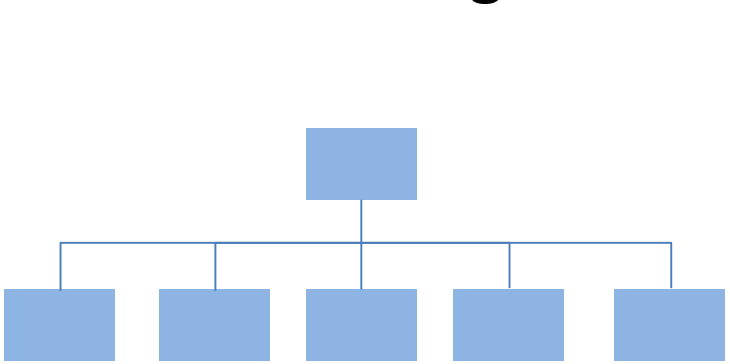
4. Support screen readers by:
   - label UI elements
   - group related content

```
<ImageButton
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_alignParentBottom="true"
   android:layout_alignParentLeft="true"
   android:layout_alignParentStart="true"
   android:contentDescription="@string/share"
   android:src="@drawable/ic_share" />
```

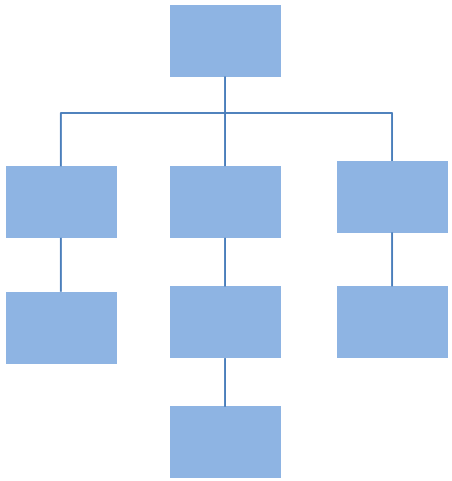
```
<ConstraintLayout
    android:id="@+id/song_data_container"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:screenReaderFocusable="true">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/song_title"
        android:text="@string/song_title" />

    ...
</ConstraintLayout>
```

Accessibility - Navigation

5. Create easy-to-follow navigation
   - Support keyboards or gestures input
   - Avoid having UI elements fade out or disappear after a certain amount of time
   - Create flat navigation structure



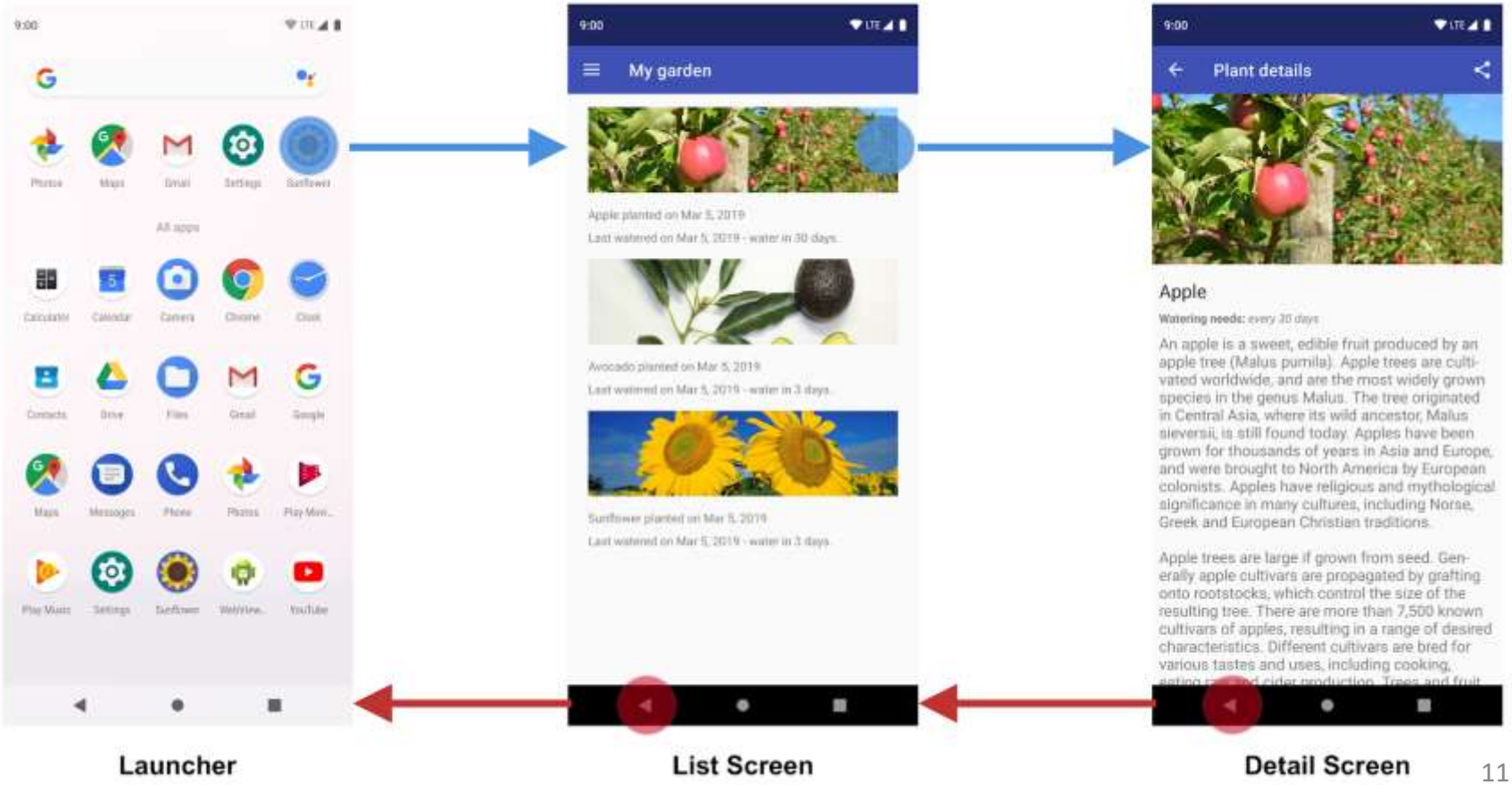
√ Do



X Don't

Accessibility - Navigation

6. Fixed start destination



Accessibility - Navigation

7. Up and Back are identical within your app's task

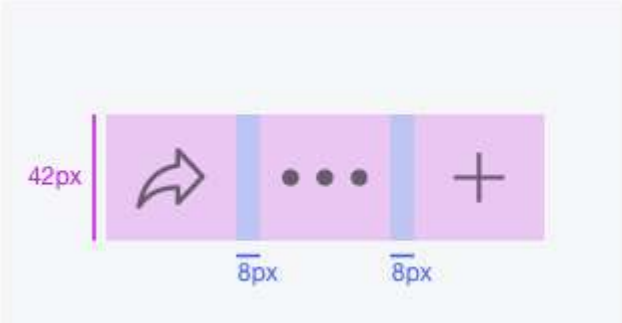
- The Up button never exists your app
- The Back button exists your app

Up and Back Buttons

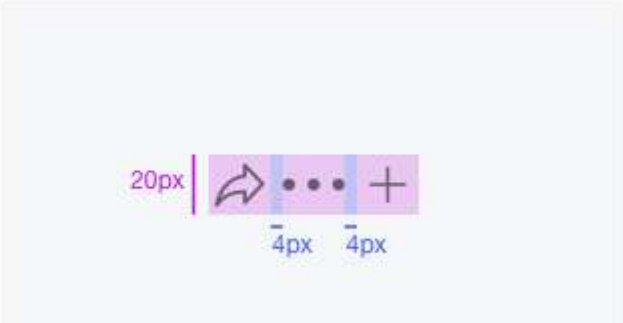
Accessibility - Navigation

8. Make touch targets large

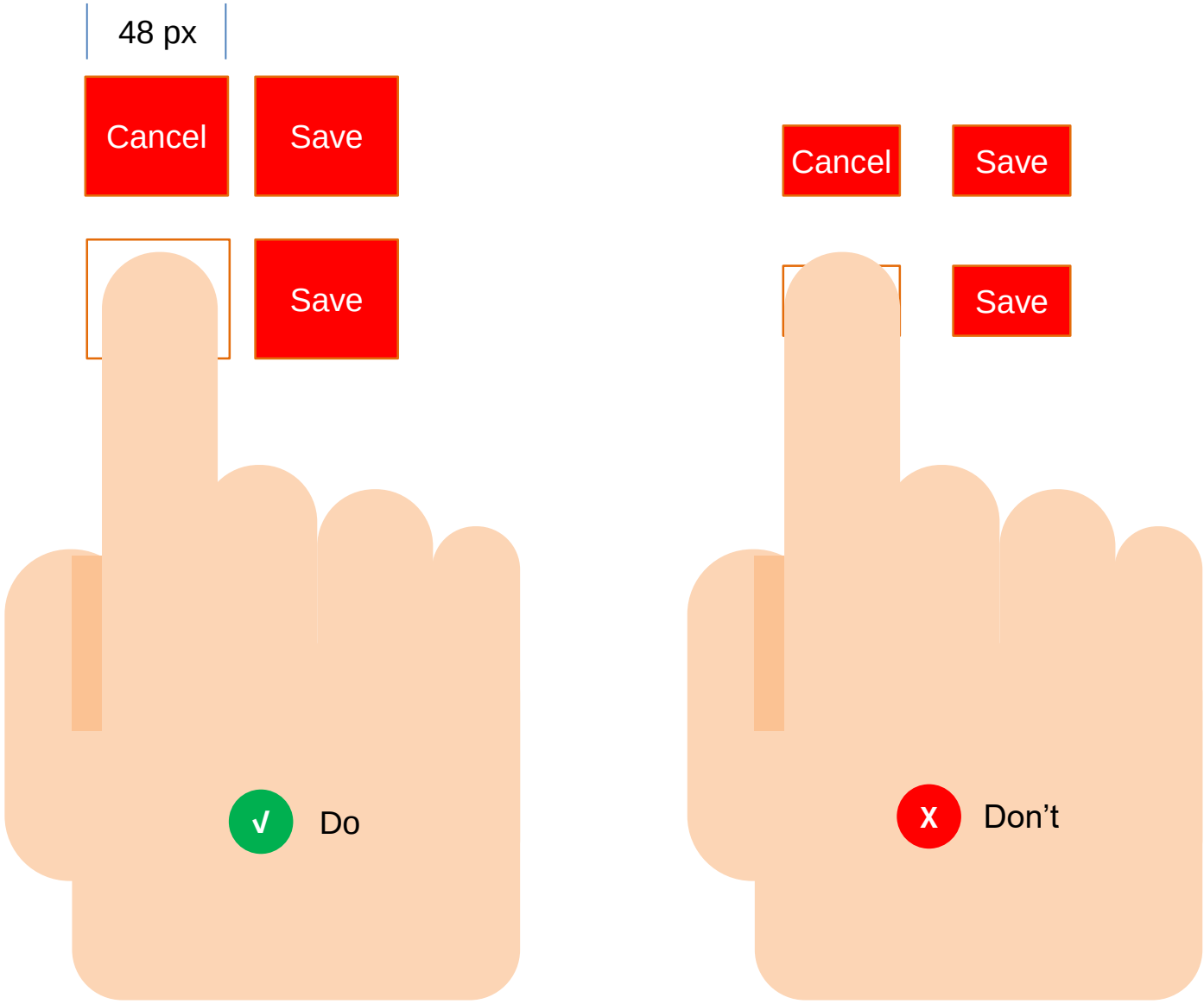
- Min size 48 x 48 pixel
- Space between elements min 8dp



√ Do



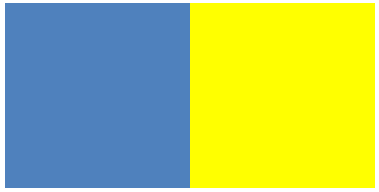
X Don't



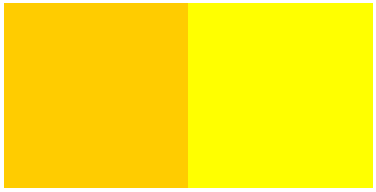
Index finger fits snugly inside.
Target edges give visual feedback.

Accessibility - Readability

1. Provide adequate color contrast

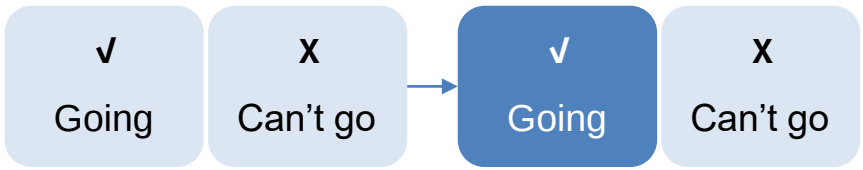


√ Do

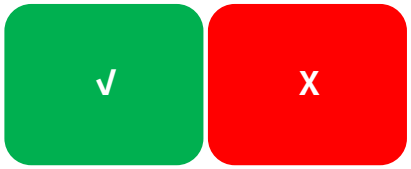


X Don't

2. Use more than just color to convey information



√ Do



X Don't

Accessibility - Readability

3. Make media content more accessible
   - Include controls for users to pause or stop video and audio files

   - Provide transcript/caption

Accessibility - Guidance and Feedback

1. Make interactive controls clear and discoverable
   - Interactive controls have text labels, tooltips, or placeholder text to indicate their purpose
   - When naming elements, be consistent in your terminology throughout your app.

2. Provide alternative text for images and video

Accessibility - Guidance and feedback

3. Offer guidance and help

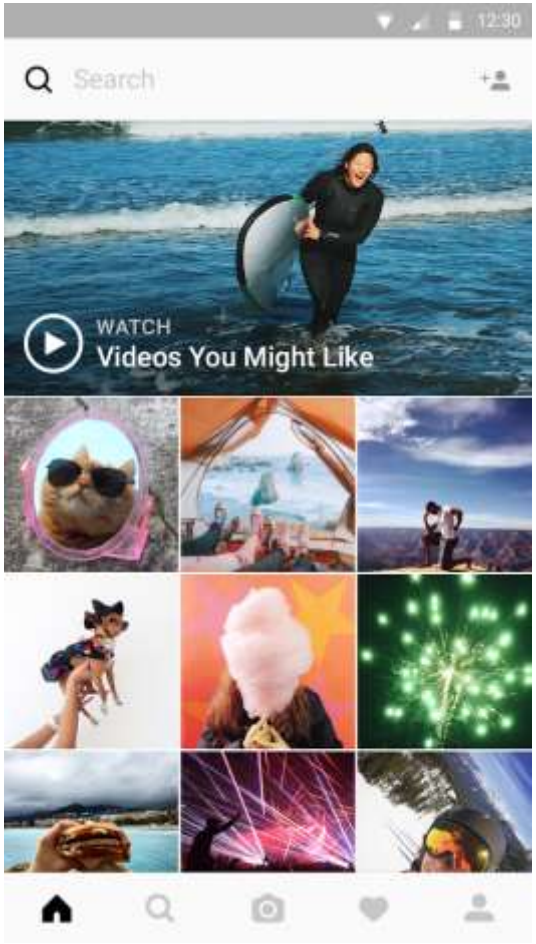
3. Give meaning to links
   - Some assistive technology modes make navigation more efficient by letting the user scan just the links, ignoring other content.
   - Generic anchor text like “click here” does not serve this purpose. A better solution is a concrete link, like “Device settings."

Question?

1. What is accessibility? Why it is important?
2. What is the main difference between navigation design for desktop and mobile app?
3. What is the main role of color in UI design?

Question?

Identify spaces allocated for content and navigation



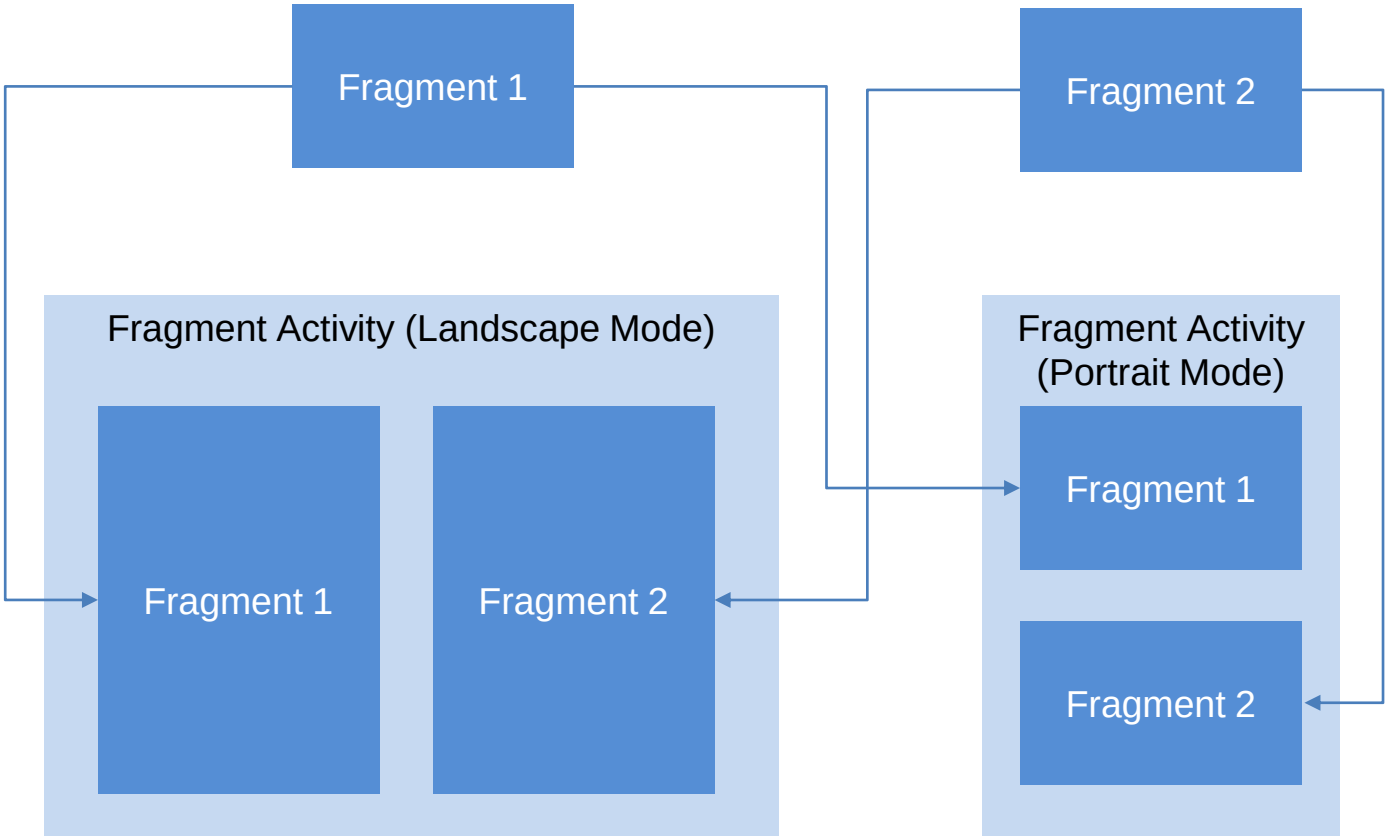
Fragment

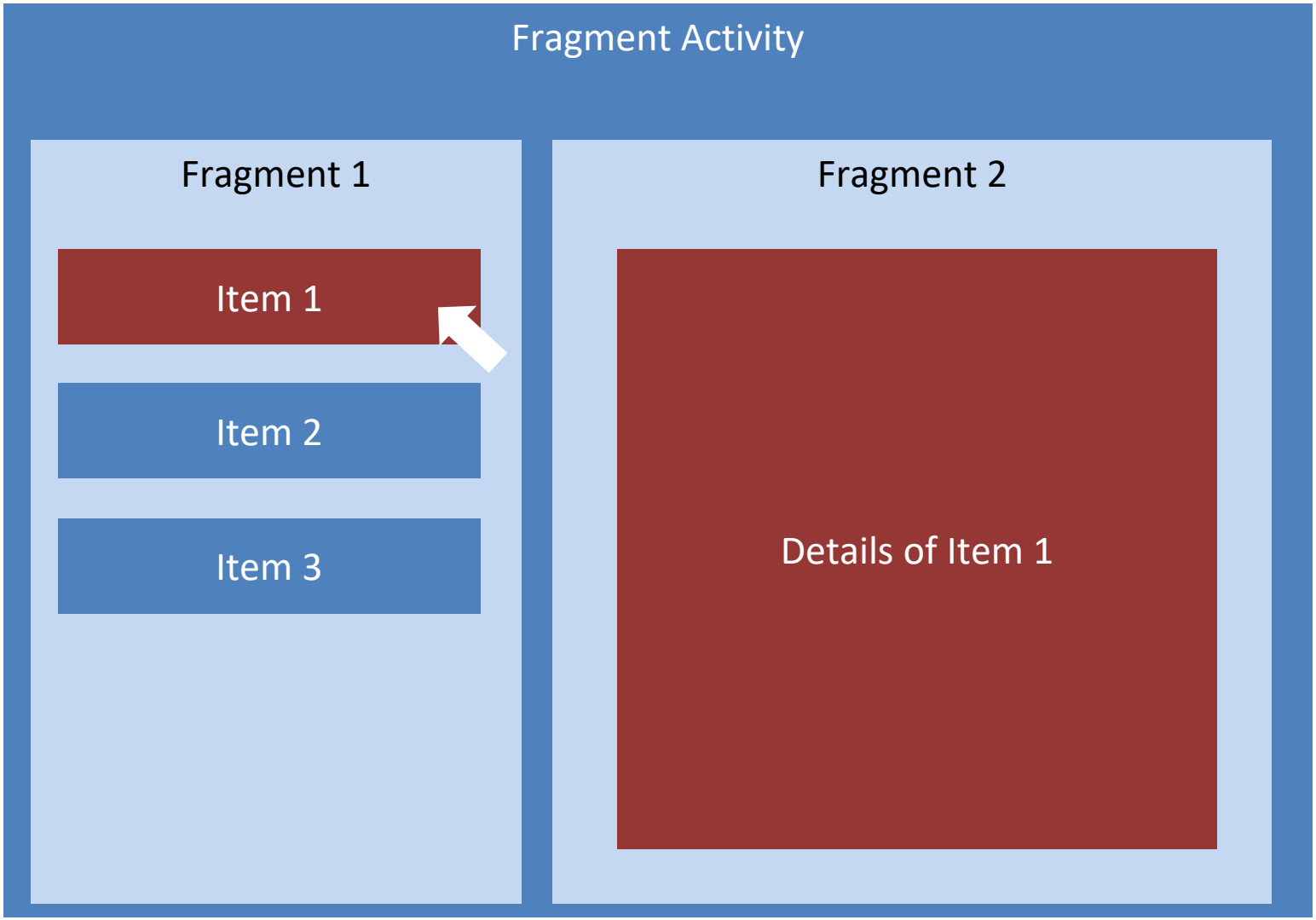
- Fragment = UI components and activity behaviours (program code)
- Each fragment is hosted by a Fragment Activity
- It can be swapped into and out of an activity

https://developer.android.com/guide/components/fragments

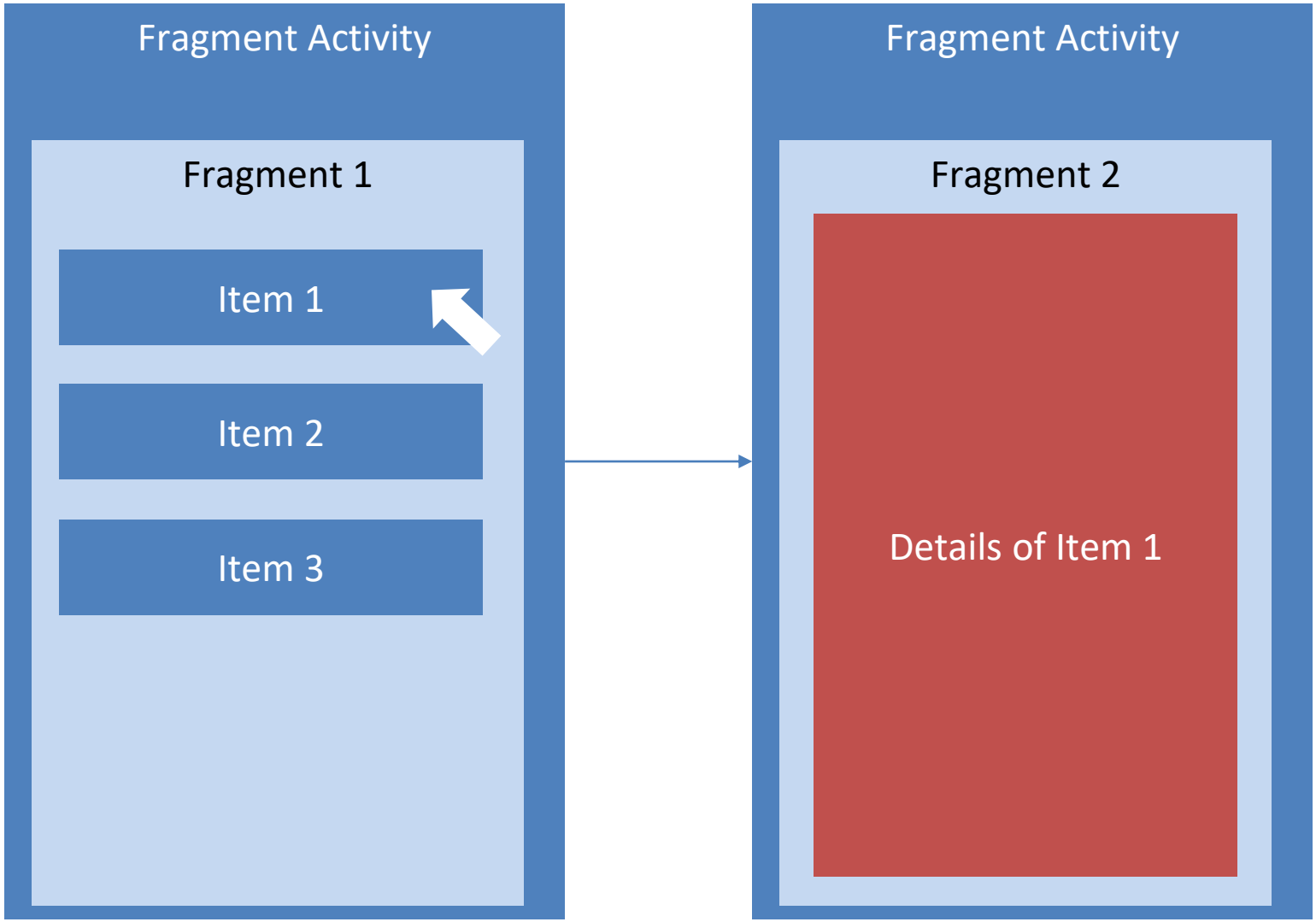
Fragment

Am example of how two UI modules defined by fragments





Landscape Mode



Portrait Mode

Creating a Fragment

Steps to include Fragment in an Activity:

1. Create a subclass of Fragment
2. Add UI (layout)
3. Manage Fragment

Step 1: Create a Fragment class

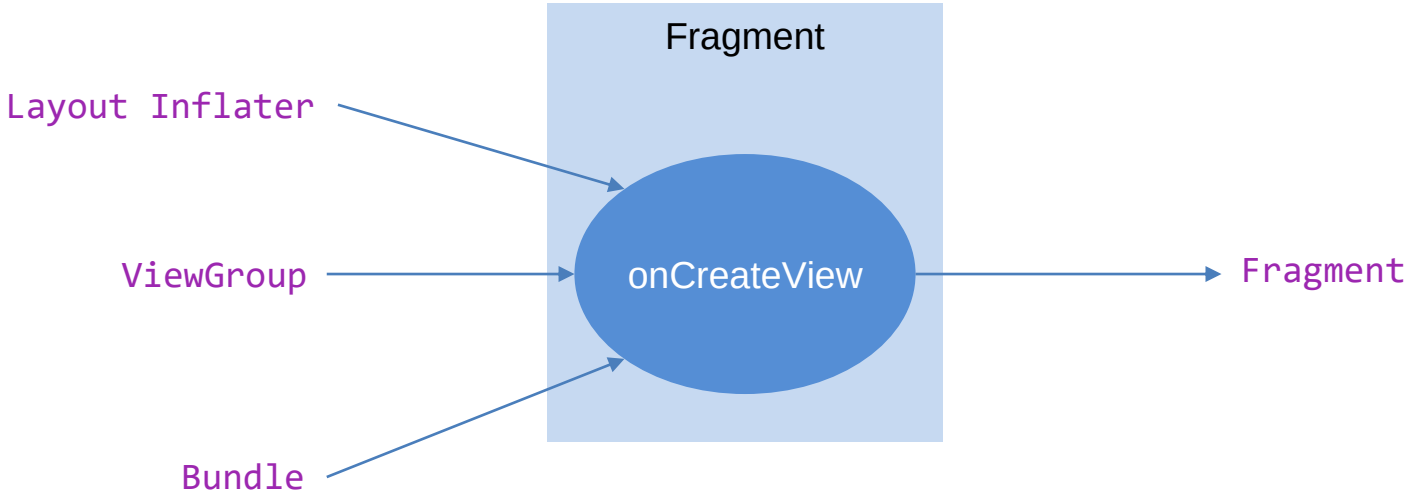
Kotlin

```kotlin
class ExampleFragment : Fragment() {

    override fun onCreateView(
            inflater: LayoutInflater,
            container: ViewGroup?,
            savedInstanceState: Bundle?
    ): View {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.example_fragment, container, false)
    }
}
```

Java

```java
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);
    }
}
```

Step 1: Create a Fragment class



Object	Description
`Layout Inflater`	An object that instantiates a layout XML file into its corresponding View
`ViewGroup`	The parent of the inflated layout
`Bundle`	Data about the previous instance of the fragment

Step 2: Add layout (UI)

- Two ways to add a fragment to the activity layout:

  1. Declare a fragment inside an activity's layout file

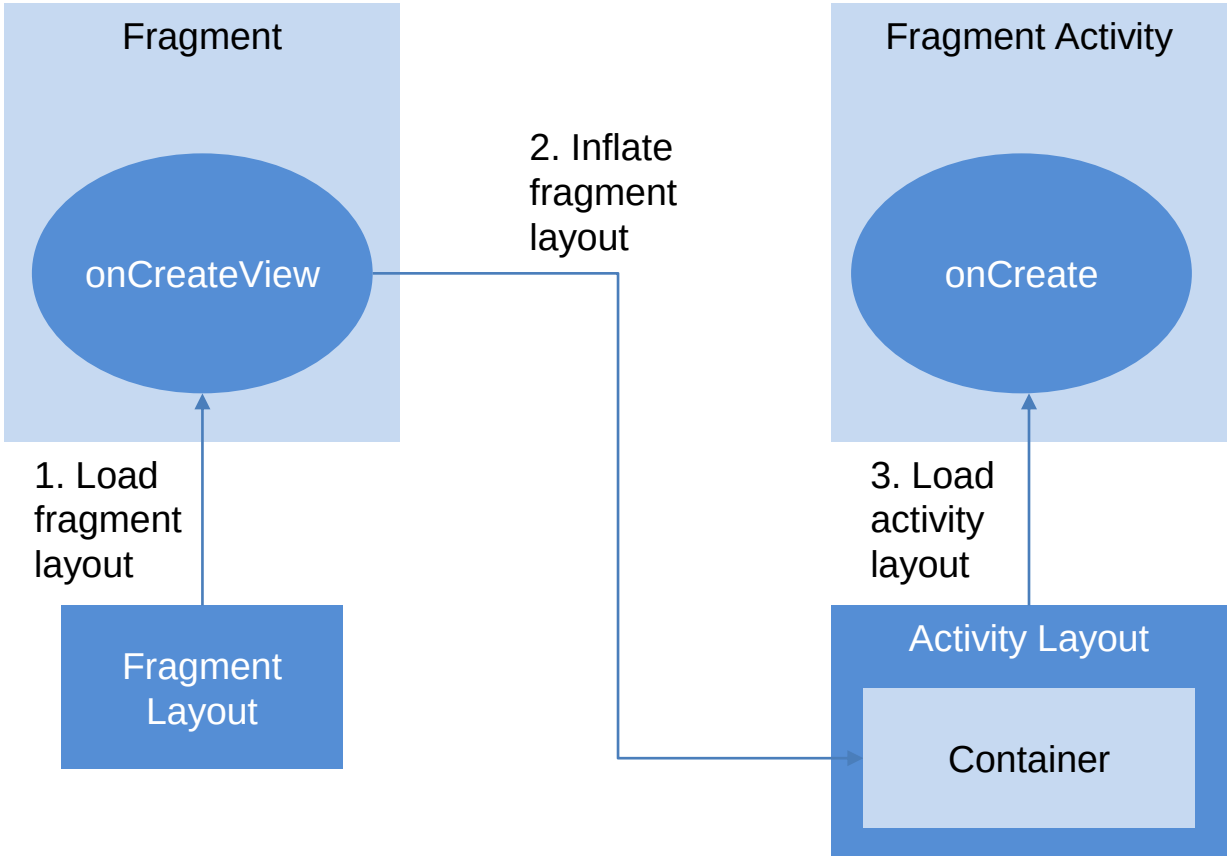
  2. Add a fragment to an activity using program

Step 2: Add layout (UI)

Method 1: Declare the fragment inside the activity's layout file

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment android:name="com.example.news.ArticleListFragment"
            android:id="@+id/list"
            android:layout_weight="1"
            android:layout_width="0dp"
            android:layout_height="match_parent" />

    <fragment android:name="com.example.news.ArticleReaderFragment"
            android:id="@+id/viewer"
            android:layout_weight="2"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
</LinearLayout>
```

Step 2: Add layout (UI)



Step 2: Add layout (UI)

Method 2: Programmatically add the fragment to an existing ViewGroup

Kotlin

```kotlin
val fragmentManager = supportFragmentManager
val fragmentTransaction = fragmentManager.beginTransaction()
val fragment = ExampleFragment()
fragmentTransaction.add(R.id.fragment_container, fragment)
fragmentTransaction.commit()
```

Java

```java
FragmentManager fragmentManager = getSupportFragmentManager();
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
ExampleFragment fragment = new ExampleFragment();
fragmentTransaction.add(R.id.fragment_container, fragment);
fragmentTransaction.commit();
```

Step 3: Manage Fragment

- Use `supportFragmentManager` to add, remove, replace fragment
- Call `addToBackStack()` so that the Back button can be used to reverse all actions
- Must call `commit()` last

Step 3: Manage Fragment

Kotlin

```kotlin
// Create new fragment and transaction
val newFragment = ExampleFragment()
val transaction = supportFragmentManager.beginTransaction()

transaction.replace(R.id.fragment_container, newFragment)

// Add the transaction to the back stack
transaction.addToBackStack(null)

transaction.commit()
```

Java

```java
// Create new fragment and transaction
Fragment newFragment = new ExampleFragment();
FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();

transaction.replace(R.id.fragment_container, newFragment);

// Add the transaction to the back stack
transaction.addToBackStack(null);

// Commit the transaction
transaction.commit();
```

Communicating with the Activity

- A Fragment can interact with the hosting Activity

Kotlin

```kotlin
// Code in a Fragment can access the hosting Activity's view
val listView: View? = activity?.findViewById(R.id.list)
```

Java

```java
// Code in a Fragment can access the hosting Activity's view
View listView = getActivity().findViewById(R.id.list);
```

Kotlin

```kotlin
// Code in an Activity can access a Fragment
val fragment = supportFragmentManager
                    .findFragmentById(R.id.example_fragment)
                        as ExampleFragment
```

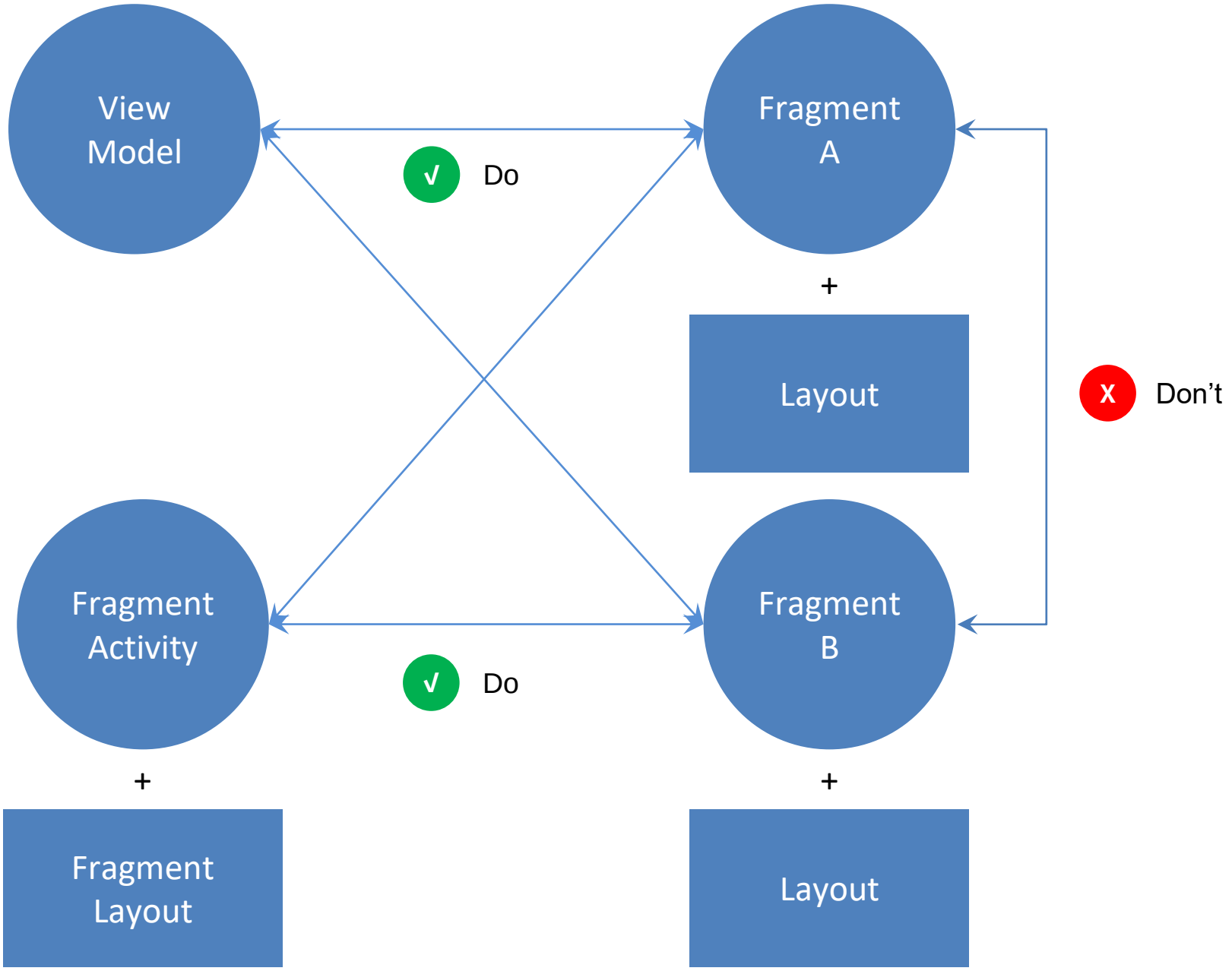
Java

```java
// Code in an Activity can access a Fragment
ExampleFragment fragment = (ExampleFragment) getSupportFragmentManager()
                                .findFragmentById(R.id.example_fragment);
```

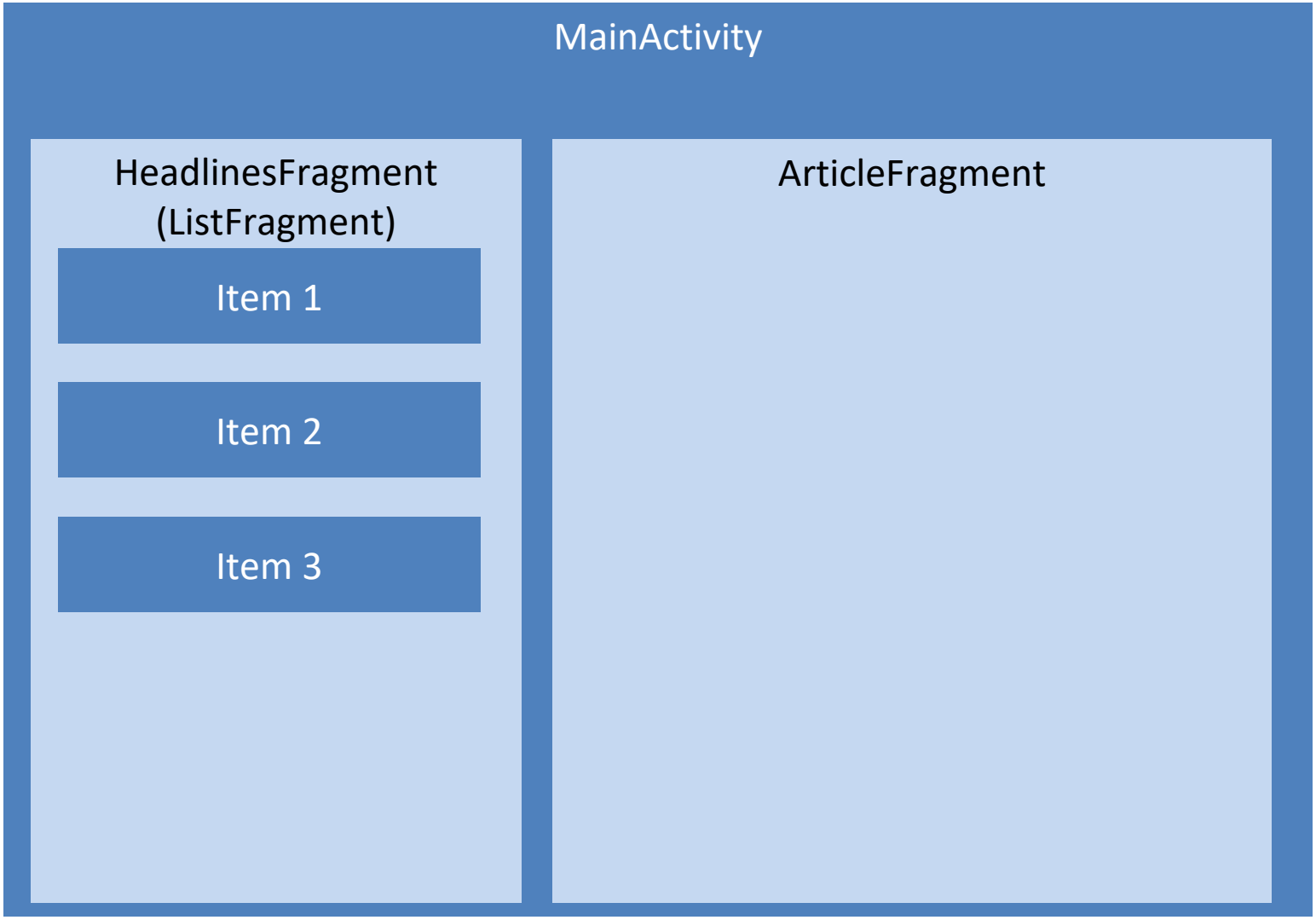
Communication between Fragments

- Two methods Fragment-to-fragment communication:
  1. A shared ViewModel (a better method compared to 2. Will be discussed in Chapter 4)
  2. Host Activity

- Two Fragments should never communication directly


View Model
Fragment A
√ Do
+
Layout
X Don't
Fragment Activity
Fragment B
√ Do
+
+
Fragment Layout
Layout

Use Case


MainActivity
HeadlinesFragment
(ListFragment)
Item 1
Item 2
Item 3
ArticleFragment

Kotlin

```kotlin
class HeadlinesFragment : ListFragment() {
    internal var callback: OnHeadlineSelectedListener

    fun setOnHeadlineSelectedListener(callback: OnHeadlineSelectedListener) {
        this.callback = callback
    }

    // This interface can be implemented by the Activity, parent Fragment,
    // or a separate test implementation.
    interface OnHeadlineSelectedListener {
        fun onArticleSelected(position: Int)
    }

    override fun onListItemClick(l: ListView, v: View, position: Int, id: Long) {
        // Send the event to the host activity
        callback.onArticleSelected(position)
    }

}
```

Kotlin

```kotlin
class MainActivity : Activity(), HeadlinesFragment.OnHeadlineSelectedListener {
    // ...

    fun onAttachFragment(fragment: Fragment) {
        if (fragment is HeadlinesFragment) {
            fragment.setOnHeadlineSelectedListener(this)
        }
    }

    fun onArticleSelected(position: Int) {
        // The user selected the headline of an article from the HeadlinesFragment
        // Do something here to display that article
    }

}
```

Kotlin

```kotlin
class MainActivity : Activity(), HeadlinesFragment.OnHeadlineSelectedListener {
    ...

    fun onArticleSelected(position: Int) {
        // The user selected the headline of an article from the HeadlinesFragment
        // Do something here to display that article
        val articleFrag = supportFragmentManager.findFragmentById(R.id.article_fragment)
                          as ArticleFragment?

        if (articleFrag != null) {
            // If article frag is available, we're in two-pane layout...
            // Call a method in the ArticleFragment to update its content
            articleFrag.updateArticleView(position)
        } else {
            // Otherwise, we're in the one-pane layout and must swap frags...
            // Create fragment and give it an argument for the selected article
            val newFragment = ArticleFragment()
            val args = Bundle()
            args.putInt(ArticleFragment.ARG_POSITION, position)
            newFragment.arguments = args

            val transaction = supportFragmentManager.beginTransaction()

            // Replace whatever is in the fragment_container view with this fragment,
            // and add the transaction to the back stack so the user can navigate back
            transaction.replace(R.id.fragment_container, newFragment)
            transaction.addToBackStack(null)
            transaction.commit()
        }
```

Java

```
public class HeadlinesFragment extends ListFragment {
    OnHeadlineSelectedListener callback;

    public void setOnHeadlineSelectedListener(OnHeadlineSelectedListener callback) {
        this.callback = callback;
    }

    // This interface can be implemented by the Activity, parent Fragment,
    // or a separate test implementation.
    public interface OnHeadlineSelectedListener {
        public void onArticleSelected(int position);
    }

    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {
        // Send the event to the host activity
        callback.onArticleSelected(position);
    }

}
```

Java

```
public static class MainActivity extends Activity
        implements HeadlinesFragment.OnHeadlineSelectedListener{
    // ...

    @Override
    public void onAttachFragment(Fragment fragment) {
        if (fragment instanceof HeadlinesFragment) {
            HeadlinesFragment headlinesFragment = (HeadlinesFragment) fragment;
            headlinesFragment.setOnHeadlineSelectedListener(this);
        }
    }

    public void onArticleSelected(int position) {
        // The user selected the headline of an article from the HeadlinesFragment
        // Do something here to display that article
    }


}
```

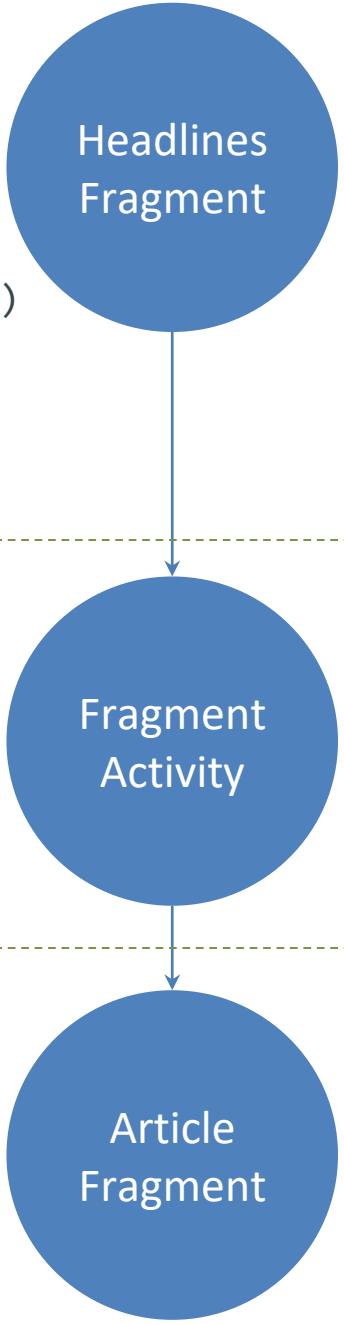
Java

```
public static class MainActivity extends Activity
        implements HeadlinesFragment.OnHeadlineSelectedListener{
    ...

    public void onArticleSelected(int position) {
        // The user selected the headline of an article from the HeadlinesFragment
        ArticleFragment articleFrag = (ArticleFragment)
                getSupportFragmentManager().findFragmentById(R.id.article_fragment);

        if (articleFrag != null) {
            // If article frag is available, we're in two-pane layout
            articleFrag.updateArticleView(position);
        } else {
            // Otherwise, we're in the one-pane layout and must swap frags
            ArticleFragment newFragment = new ArticleFragment();
            Bundle args = new Bundle();
            args.putInt(ArticleFragment.ARG_POSITION, position);
            newFragment.setArguments(args);

            FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();

            // Replace whatever is in the fragment_container view with this fragment,
            // and add the transaction to the back stack so the user can navigate back
            transaction.replace(R.id.fragment_container, newFragment);
            transaction.addToBackStack(null);

            // Commit the transaction
            transaction.commit();
        }
    }
```

```
OnHeadlineSelectedListener callback

public interface OnHeadlineSelectedListener {
     public void onArticleSelected(int position);
}
@Override
public void onListItemClick(ListView l, View v, int position, long id)
{
     // Send the event to the host activity
     callback.onArticleSelected(position);
}
```

```
public void onArticleSelected(int position) {
    ...
}

public void setArticle(int position) {
    ...
    ArticleFragment.updateArticleView(position)
}
```

```
public void updateArticleView(int position) {
  . . .
}
```



Question?

1. What is the main difference between a Fragment and an Activity?
2. What are the advantages of using Fragment compared to Activity?
3. Why a fragment should not communicate with another fragment directly?

Dialogs

- Dialogs inform users about critical information, require users to make decisions, or encapsulate multiple tasks within a discrete process.

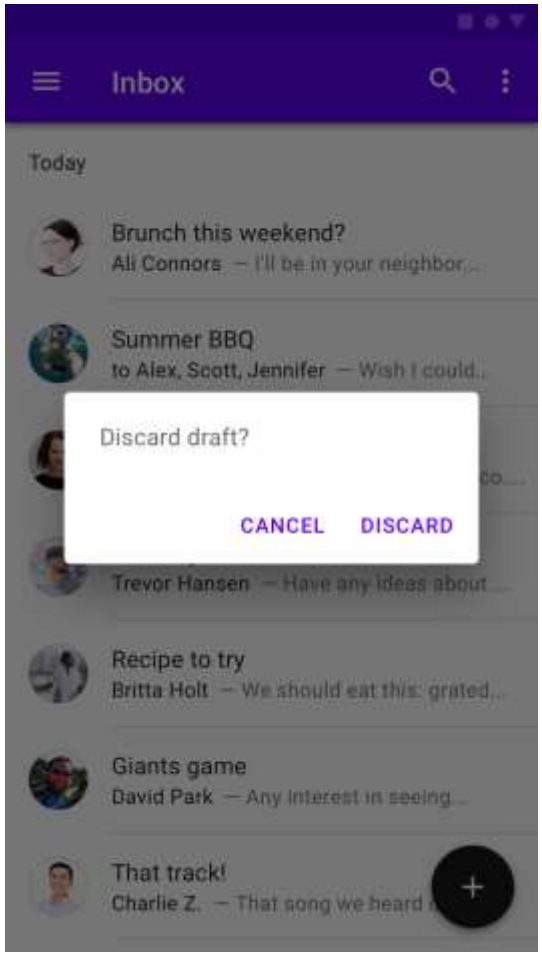
- Use dialogs sparingly because they are interruptive in nature.

https://developer.android.com/guide/topics/ui/dialogs

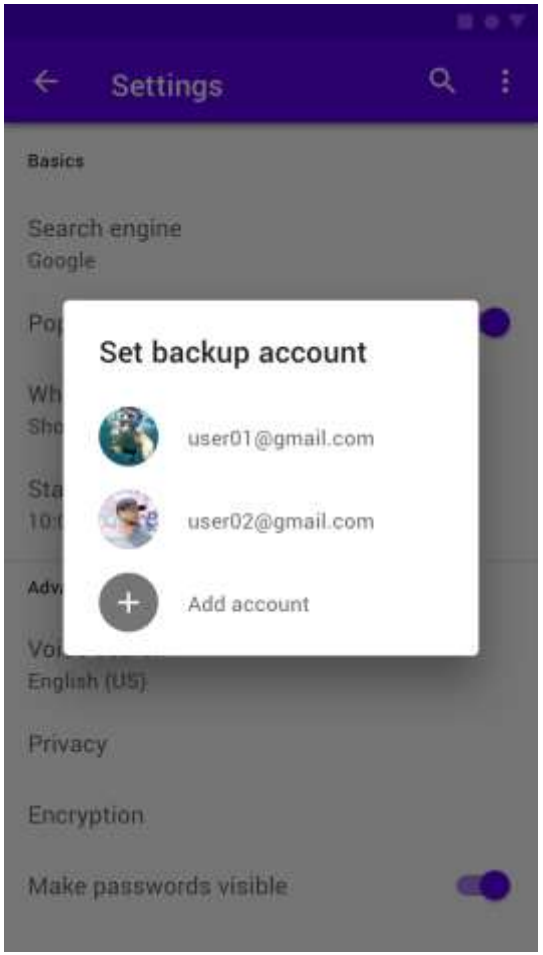
Dialog vs Notification and Snackbar

Component	Priority	User Action
Snackbar	Low	Optional It disappears automatically
Notification	Medium	Optional It remains until dismissed by the user, or if the state that caused the officiation is resolved
Dialog	High	Required It blocks app usage until the user takes a dialog action or exists the dialog

Dialogs



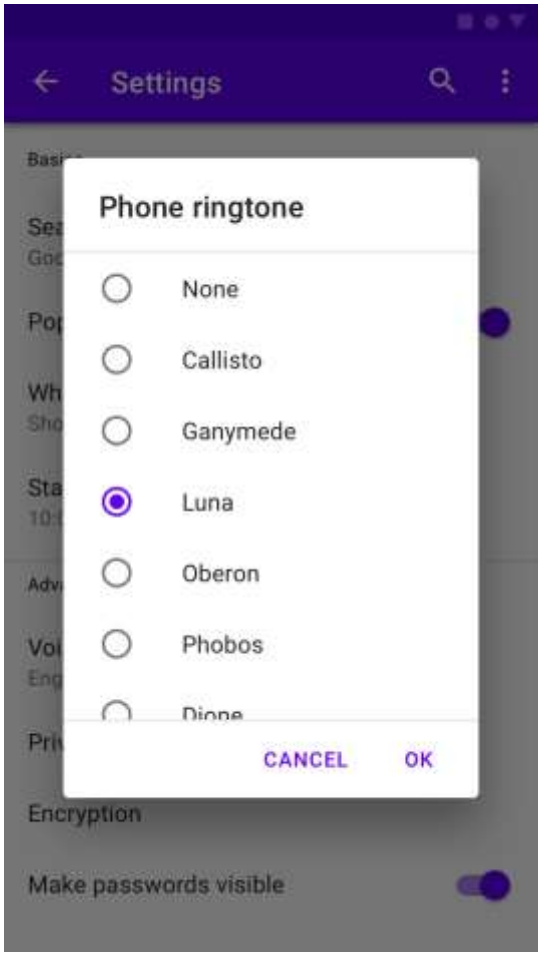
1. Alert Dialog



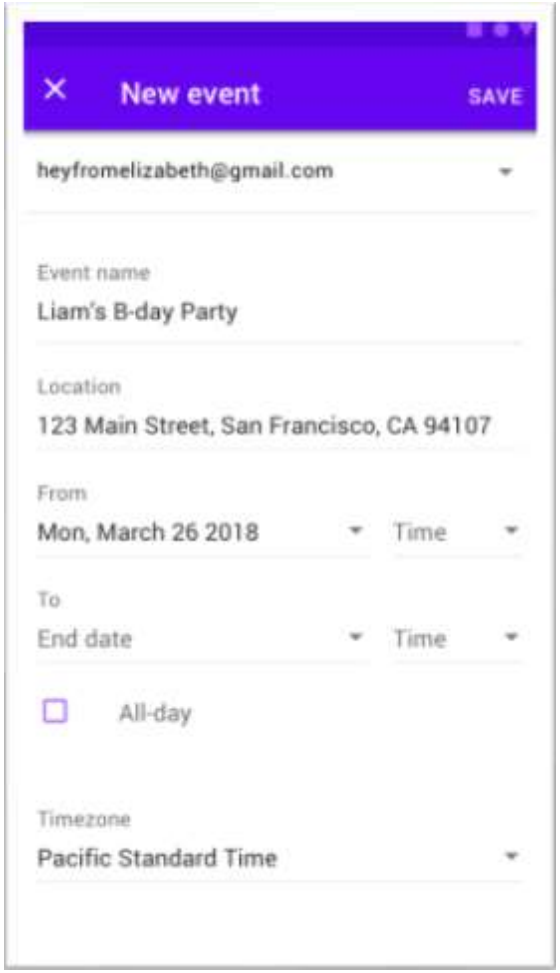
2. Simple Dialog

https://material.io/components/dialogs/

Dialogs



3. Confirmation Dialog



4. Full screen Dialog

https://material.io/components/dialogs/

Dialogs

- Avoid dialogs that:
  - Open additional dialogs
  - Contain scrolling content, particularly alerts

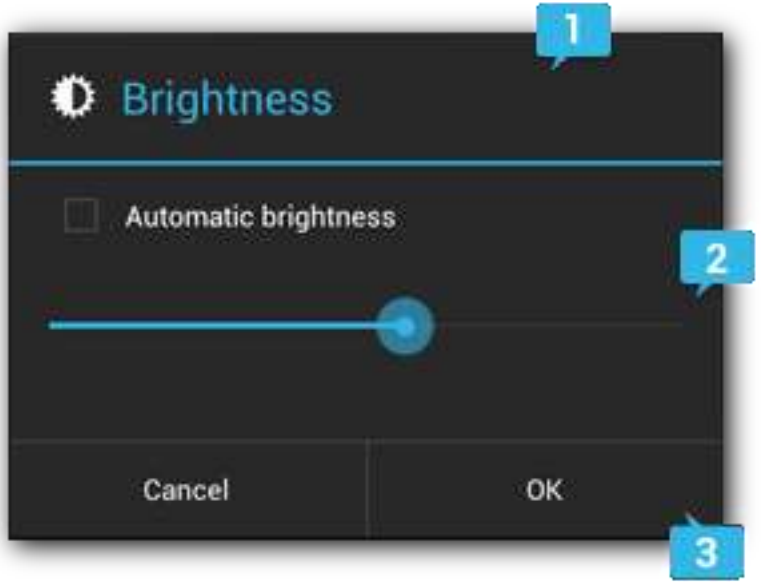
- Exceptions include:
  - Full-screen dialogs may open additional dialogs, such as pickers, because their design accommodates additional layers of material without significantly increasing the app's perceived z-depth or visual noise.

AlertDialog

1. Title (optional)

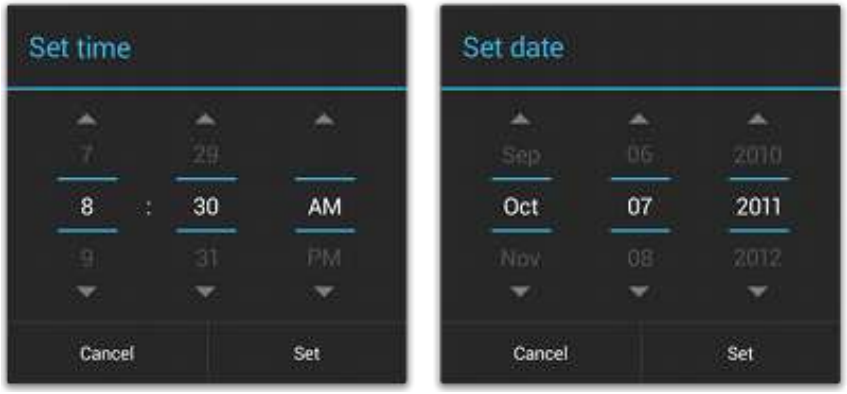
2. Content area: This can display a message, a list, or other custom layout.

3. Action buttons: There should be no more than three action buttons in a dialog.



Pickers

- Android provides controls for the user to pick a time or pick a date as ready-to-use dialogs.



- Each picker provides controls for selecting each part of the time (hour, minute, AM/PM) or date (month, day, year).

https://developer.android.com/guide/topics/ui/controls/pickers

Pickers

- Use DialogFragment to host each time or date picker.

- The DialogFragment manages the dialog lifecycle for you and allows you to display the pickers in different layout configurations

Pickers

- DialogFragment was first added to the platform in Android 3.0 (API level 11)

- If your app supports versions of Android older than 3.0—even as low as Android 1.6—you can use the DialogFragment class that's available in the support library for backward compatibility.

Pickers

- To define a DialogFragment for a TimePickerDialog, you must:
  - Define the onCreateDialog() method to return an instance of TimePickerDialog
  - Implement the TimePickerDialog.OnTimeSetListener interface to receive a callback when the user sets the time.

Pickers

Kotlin

```
class TimePickerFragment : DialogFragment(), TimePickerDialog.OnTimeSetListener {

    override fun onCreateDialog(savedInstanceState: Bundle): Dialog {
        // Use the current time as the default values for the picker
        val c = Calendar.getInstance()
        val hour = c.get(Calendar.HOUR_OF_DAY)
        val minute = c.get(Calendar.MINUTE)

        // Create a new instance of TimePickerDialog and return it
        return TimePickerDialog(activity, this, hour, minute,
                                DateFormat.is24HourFormat(activity))
    }

    override fun onTimeSet(view: TimePicker, hourOfDay: Int, minute: Int) {
        // Do something with the time chosen by the user
    }
}
```

Pickers

Java

```
public static class TimePickerFragment extends DialogFragment
                            implements TimePickerDialog.OnTimeSetListener {

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the current time as the default values for the picker
        final Calendar c = Calendar.getInstance();
        int hour = c.get(Calendar.HOUR_OF_DAY);
        int minute = c.get(Calendar.MINUTE);

        // Create a new instance of TimePickerDialog and return it
        return new TimePickerDialog(getActivity(), this, hour, minute,
                DateFormat.is24HourFormat(getActivity()));
    }

    public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
        // Do something with the time chosen by the user
    }
}
```

Picker

In the layout file and Activity class:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/pick_time"
    android:onClick="showTimePickerDialog" />
```

Kotlin

```
fun showTimePickerDialog(v: View) {
    TimePickerFragment().show(supportFragmentManager, "timePicker")
}
```

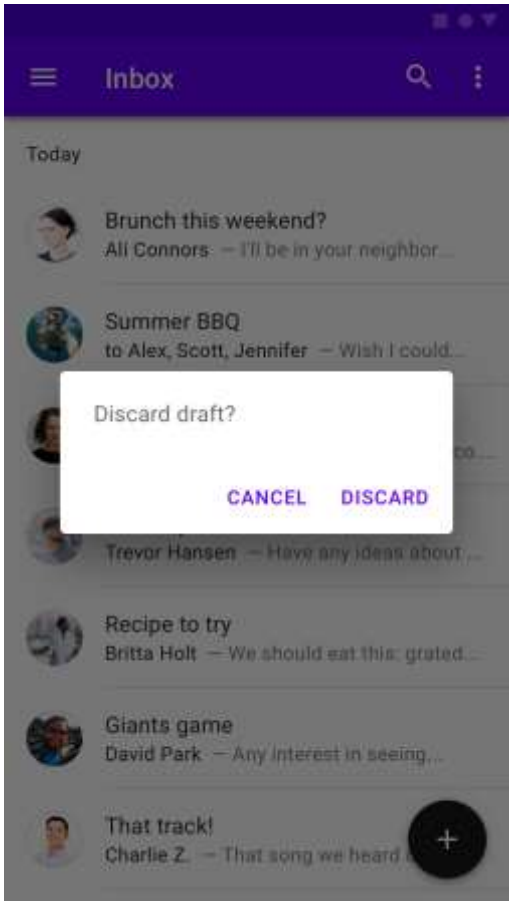
Java

```
public void showTimePickerDialog(View v) {
    DialogFragment newFragment = new TimePickerFragment();
    newFragment.show(getSupportFragmentManager(), "timePicker");
}
```

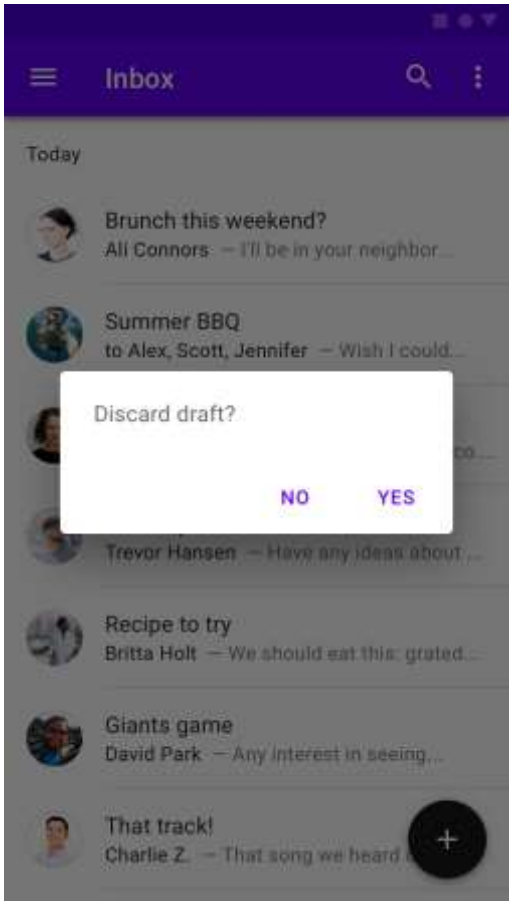
Question?

1. Which AlertDialog is better? Why?

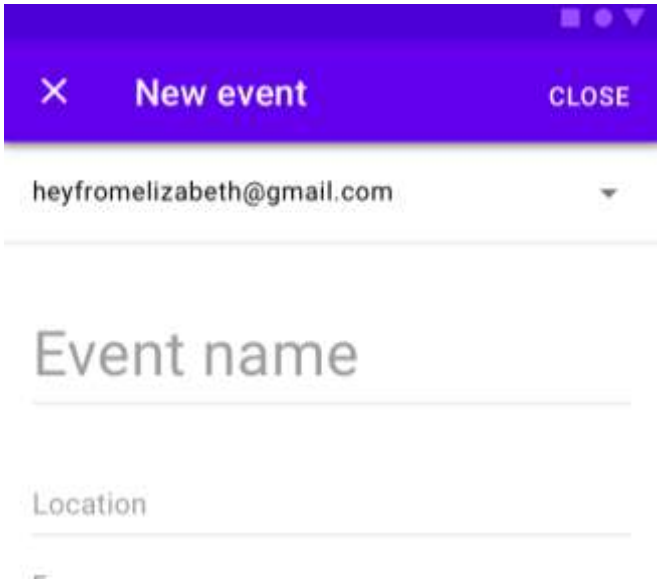
A



B

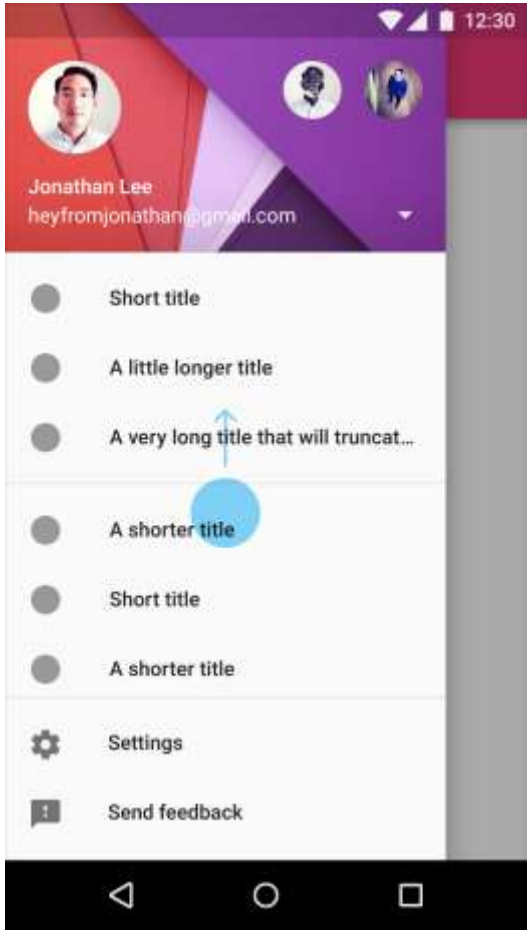


2. Evaluate the following full screen dialog:



Navigation Drawer

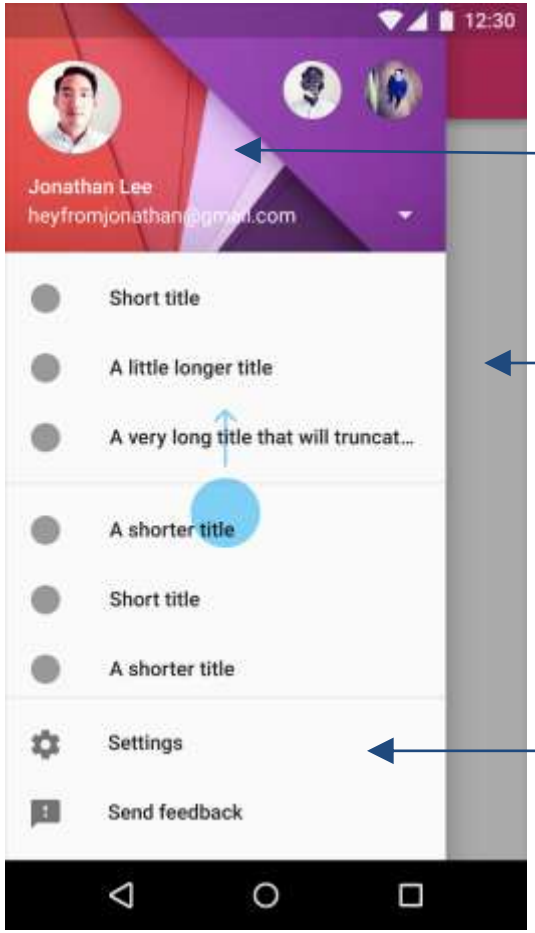
- Displays the app's main navigation options
- Behaviour:
  - show when the user swipes a finger from the left edge of the screen or,
  - user touches the app icon in the action bar



https://material.io/components/navigation-drawer/

Navigation Drawer

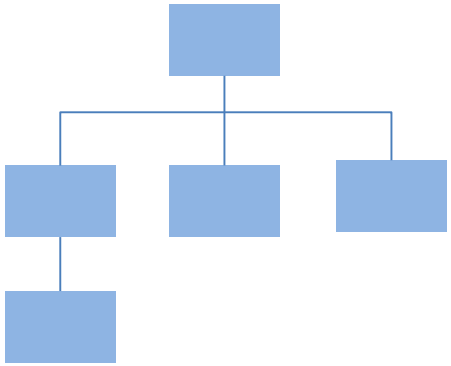
- Implemented using DrawerLayout
- Populated using Fragment - FrameLayout



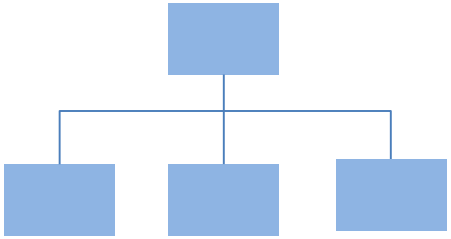
Navigation Drawer

- Recommended for:

>= 5 top-level destinations



>= 2 levels of navigation hierarchy



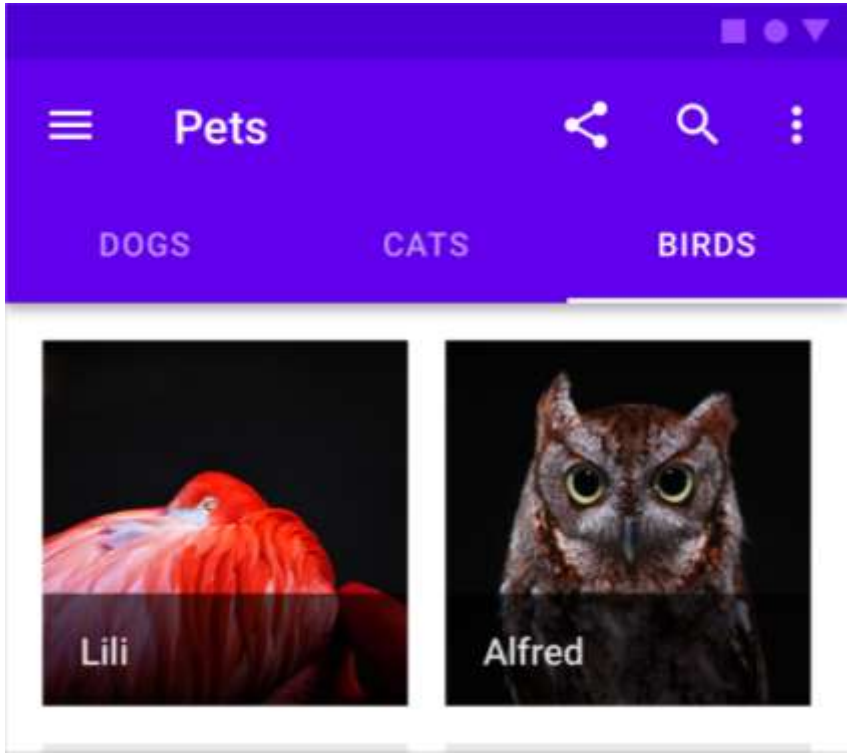
Quick navigation between unrelated destinations

Tabs

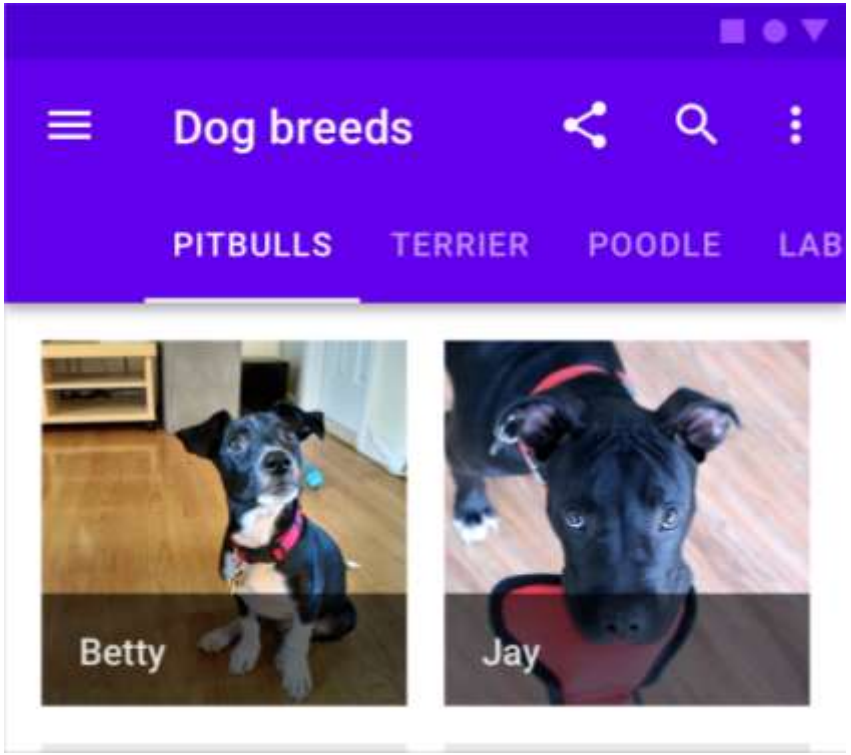
- It organizes and allows navigation between groups of content
- Content should be related and at the same level of hierarchy

https://material.io/components/tabs/#

Tabs



1. Fixed Tabs



2. Scrollable Tabs

Question?

1. Compare Navigation Drawer and Tabs
2. Among Navigation Drawer and Tabs, which UI is suitable for each of the following apps:
   a. A news app
   b. A weather app
   c. A contact app