Programming with Android: App Guidelines part 2: UI Navigation

Federico Montori

Dipartimento di Informatica: Scienza e Ingegneria

Università di Bologna

Outline

Menu: outline

- It appears whenever the user presses the menu button
- Useful for giving different options without leaving the current Activity
- Don’t make too big menus, or they’ll cover entirely the Activity

Menu: creating a menu

- Two methods (again):
  - XML
    - Place a file inside res/menu/
    - Inflate the menu inside the Activity
    - Useful if you want to create the same menu inside different activities
  - Java
    - Create the menu directly inside the activity

Menu: the **declarative** approach

- Create res/menu/menu.xml
- We need:
  - IDs of menu elements
  - Title of each element
  - Icon of each element
- Inside the Activity, create onCreateOptionsMenu()
  - Inflate the menu
  - Add functionality to the buttons

Menu: menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@+id/item1" android:title="First Option"></item>
    <item android:id="@+id/item2" android:title="Second Option">
        <menu>
            <item android:id="@+id/item3" android:title="Third Option"/>
            <item android:id="@+id/item4" android:title="Fourth Option"/>
        </menu>
    </item>
</menu>
```

Menu: inflate the menu

- Override Activity methods:

```
public boolean onCreateOptionsMenu(Menu menu) {
      super.onCreateOptionsMenu(menu);

      getMenuInflater().inflate(R.menu.myMenu, menu);

      // If you want to fire an intent when “menu_first” is pressed
      menu.findItem(R.id.menu_first).setIntent(new Intent(this, First.class));

      return true;
}
```

Menu: specify the behavior

- Override Activity methods:

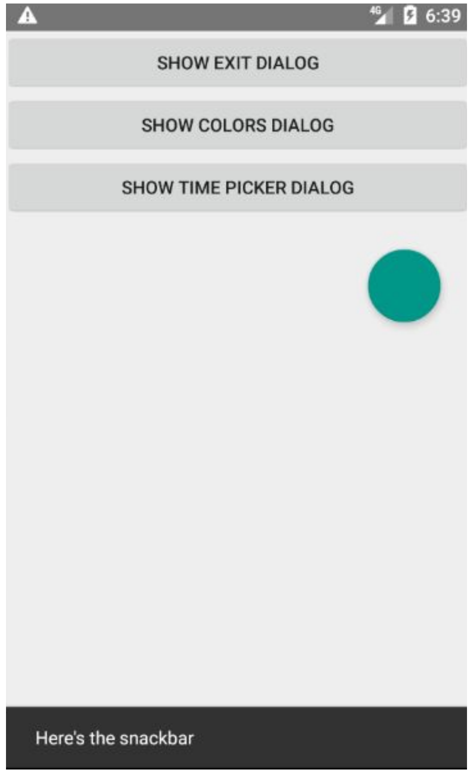
```
public boolean onOptionsItemSelected(MenuItem item) {
      switch ( item.getItemId() ) {
            case R.id.item1:
                  /* do stuff */
                  return true;
            [ … ]
            default:
                  return super.onOptionsItemSelected(item);
      }
}
```

Snackbar

- Similar to a Toast, but
  - Is attached to a view that'll hold its presence...
  - Can listen to events (mostly clicks or swipes)
  - Can declare actions to be performed
- If attached to a CoordinatorLayout gains other features
  - Can be swiped away
  - The layout handles interaction with other views

    e.g. Move the FAB
- Often attached to a FloatingActionButton
- Create it with:

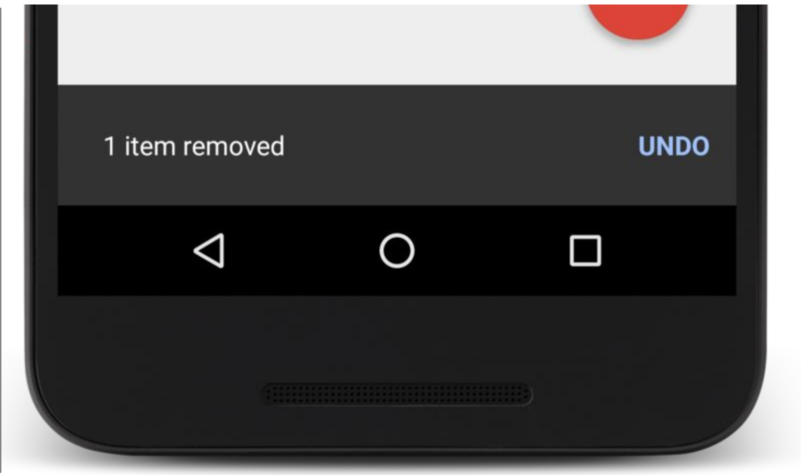
```
Snackbar.make(view, "Here's the snackbar", Snackbar.LENGTH_LONG).show()
```



Snackbar: actions

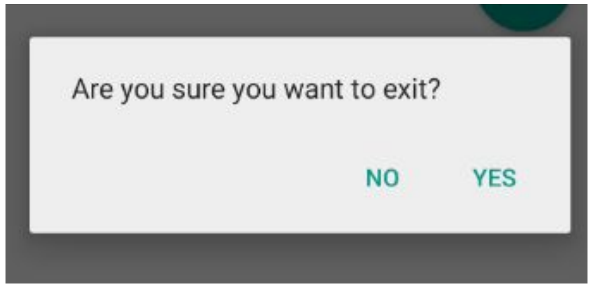
- Snackbars can also have actions in them
  - To add further options on the action just performed
  - To undo operations
  - Action must be only one, if you add more they'll overwrite
- Before calling show(), add .setAction()
  - First parameter: String to be displayed
  - Second parameter: listener that has to handle the action

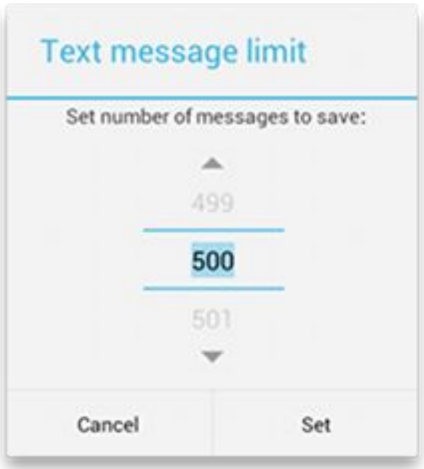
```
Snackbar.make(view, "1 item removed", Snackbar.LENGTH_LONG)
        .setAction("UNDO", new View.OnClickListener() {
            @Override
            public void onClick(View v) {  /* Replace with your action */
                Toast.makeText(getApplicationContext(),
                        "Oh my god you pressed it!",
                        Toast.LENGTH_LONG).show();
            }
        }).show();
```



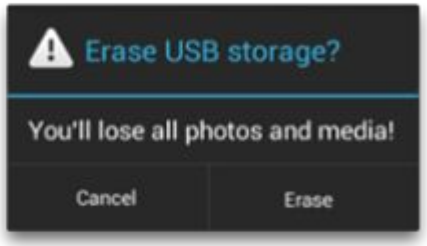
Dialog: outline

- Used to interact with the user
- Little messages, easy answers
- Different kinds:
  - AlertDialog
  - DatePickerDialog
  - TimePickerDialog

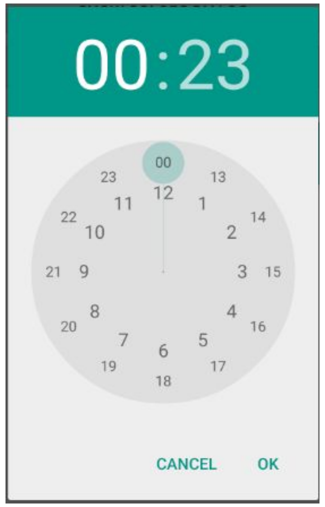

Are you sure you want to exit?

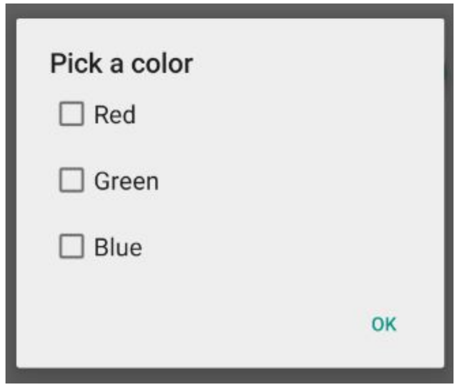

Text message limit

Set number of messages to save:


Erase USB storage?

You'll lose all photos and media!

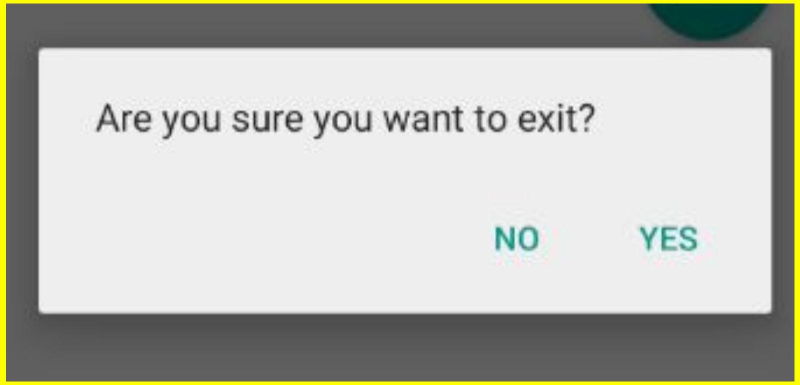



Pick a color

Dialog: AlertDialog

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setMessage("Are you sure you want to exit?").setCancelable(false);
builder.setPositiveButton("Yes", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
            MenuExampleActivity.this.finish();
    }
});
builder.setNegativeButton("No", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        dialog.cancel();
    }
});
AlertDialog alert = builder.create();       alert.show();
```

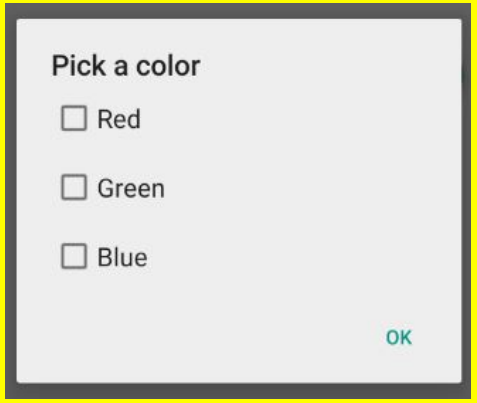
Cancelable through back?



Are you sure you want to exit?

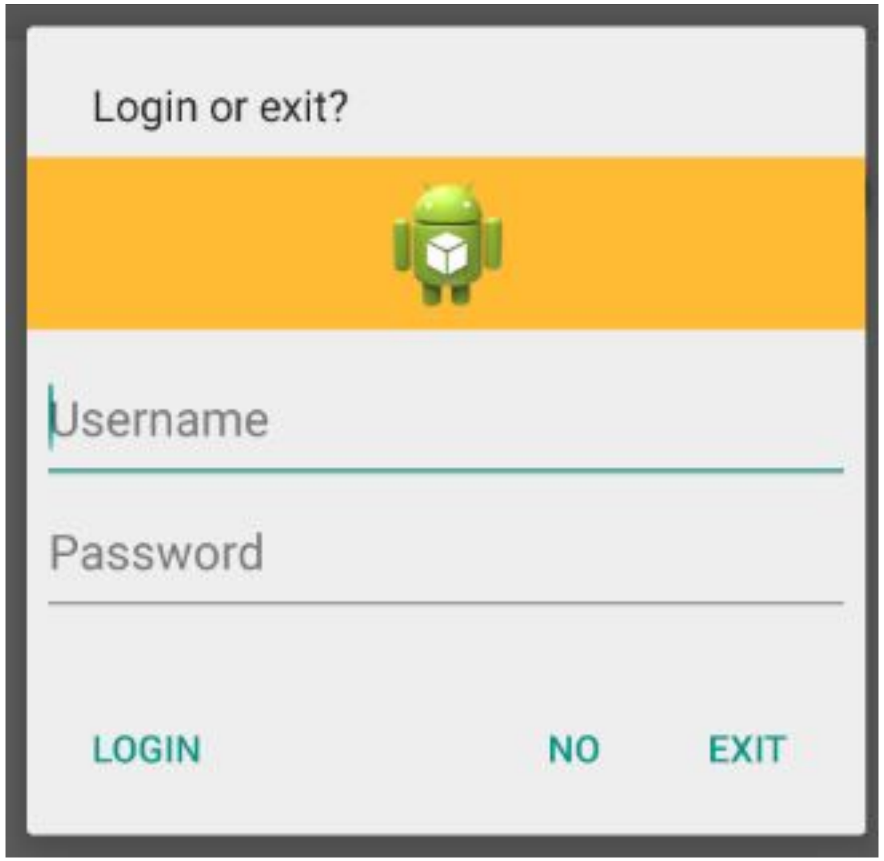
Dialog: AlertDialog with a list

```
final CharSequence[] items = {"Red", "Green", "Blue"};
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setTitle("Pick a color");
builder.setItems(items, new DialogInterface.OnClickListener() {
   public void onClick(DialogInterface dialog, int item) {
            Toast.makeText(getApplicationContext(), items[item],
            Toast.LENGTH_SHORT).show();
        }
     });// OR
builder.setSingleChoiceItems(items, -1, new DialogInterface.OnClickListener() {
   public void onClick(DialogInterface dialog, int item) {
       Toast.makeText(getApplicationContext(), items[item],
     Toast.LENGTH_SHORT).show();
   }
});
AlertDialog alert = builder.create();
```



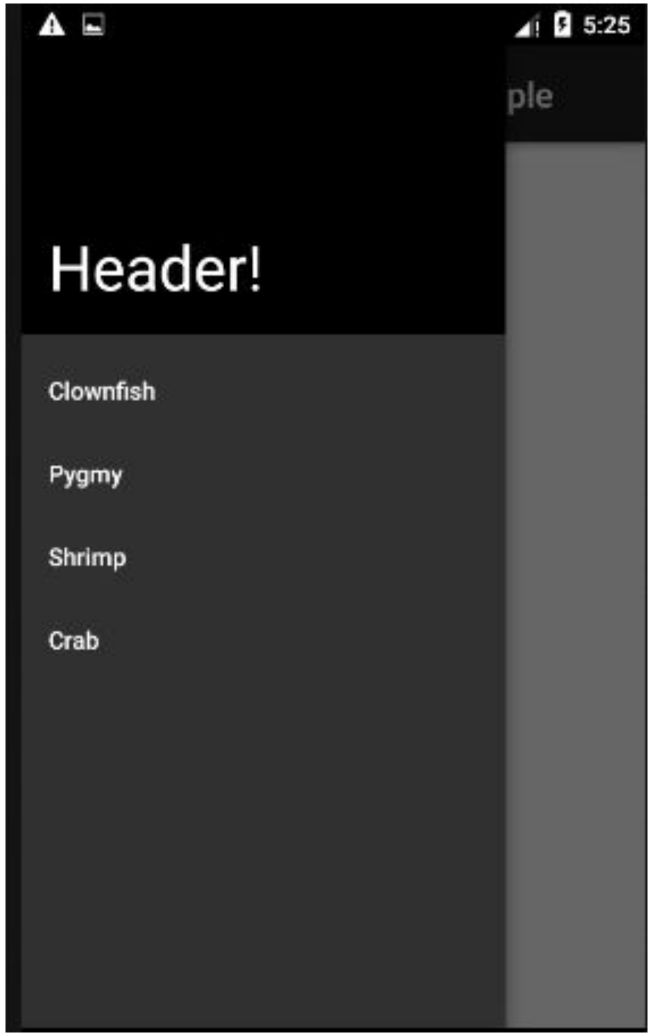
AlertDialog with custom layout

- Simply call setView() on the builder
  - Provide a suitable layout
  - Remember you can add a maximum of 3 buttons
- Implement listeners and act accordingly
- For many Dialogs you should extend a FragmentDialog instead (e.g. Listeners)



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

NavigationDrawer



- Novel navigation component
- Hidden when not in use, appears when swiping from the left or by clicking on the top-left drawer icon

- Add proper dependencies for older versions of SDK

```
dependencies {
  implementation 'com.android.support:appcompat-v7:27.1.0'
  implementation 'com.android.support:design:27.1.0'
}
```

Adding a NavigationDrawer

- Should be added as root view inside the layout
  - It has to contain two items
    - Layout when NavigationDrawer is hidden (YourMainLayout)
    - Content of the navigation drawer (similar to a menu)

```
<androidx.drawerlayout.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android">
    <YourMainLayout …> … </YourMainLayout>

    <com.google.android.material.navigation.NavigationView
            …
        app:headerLayout="@layout/MyHeader"
        app:menu="@menu/myMenu"
            … />
</androidx.drawerlayout.widget.DrawerLayout>
```

Only valid for AndroidX
Otherwise the syntax is a bit different

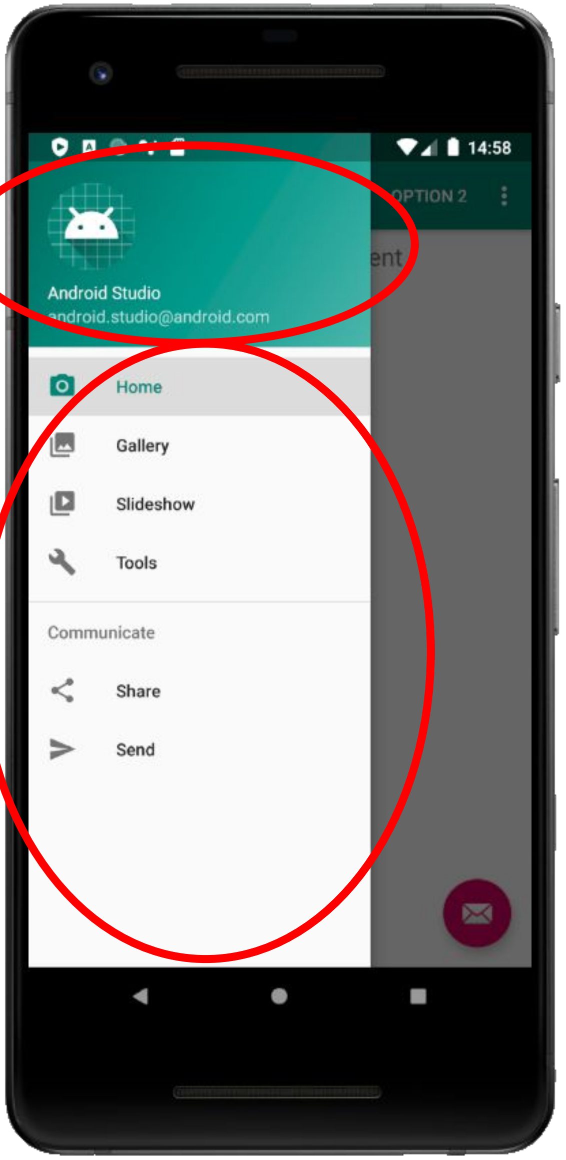
Defining content and header

res/layout/myHeader.xml

```
<LinearLayout …>
   <ImageView … /> <TextView … /> <TextView … />
</LinearLayout>
```

res/menu/myMenu.xml

```
<menu
      xmlns:android="http://schemas.android.com/apk/res/android">
      <group android:checkableBehavior="single">
            <item … />  <item … />  <item … />  <item … />
      </group>
      <item android:title="Communicate">
            <menu>
                  <item … />  <item … />
            </menu> </item>
</menu>
```



Listening to **events**

- As many other Android Components, NavigationDrawer fires events as well

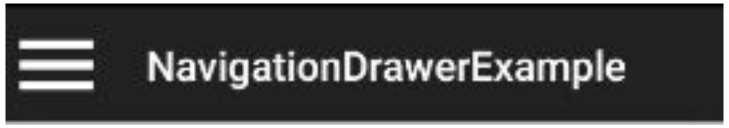
```
NavigationView navigationView = findViewById(R.id.nav_view);

navigationView.setNavigationItemSelectedListener(
        new NavigationView.OnNavigationItemSelectedListener() {
          @Override
          public boolean onNavigationItemSelected(MenuItem menuItem) {
            menuItem.setChecked(true);
            mDrawerLayout.closeDrawers();
            ...
            return true;
          }
        });
```

This is a very simple way…
- Use Navigation Framework
- NavController
- NavigationUI

Adding a toolbar

- Not mandatory, as the NavigationDrawer still answers to swipe events

- … but …



- It tells your user that there is more content to see
- Also provides an alternative to access content
- It meets the Material Design guidelines

Adding a toolbar

- Add the following inside the layout

```
<com.google.android.material.appbar.AppBarLayout android:theme="@style/AppTheme.AppBarOverlay">

   <androidx.appcompat.widget.Toolbar app:popupTheme="@style/AppTheme.PopupOverlay" />
</com.google.android.material.appbar.AppBarLayout>
```

- Set an appropriate theme in AndroidManifest.xml

```
android:theme="@style/AppTheme"
```

- And in the Java class

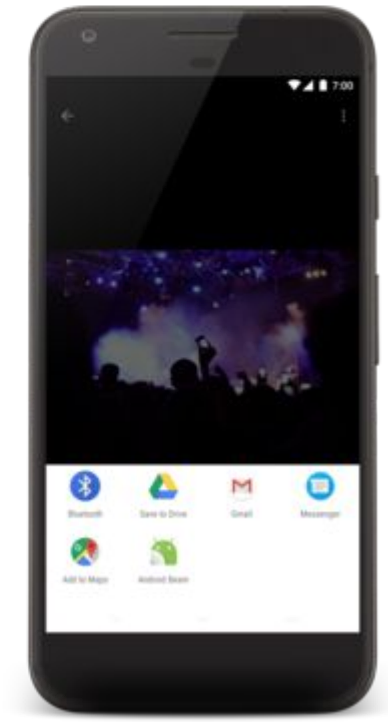
```
Toolbar toolbar = findViewById(R.id.myToolbar);
setSupportActionBar(toolbar);
ActionBar actionbar = getSupportActionBar();
actionbar.setDisplayHomeAsUpEnabled(true);
actionbar.setHomeAsUpIndicator(R.drawable.ic_menu);
```

Stuff About the home icon…
in the next slide how to call it back.

Sharing data (even easier!)

- Starting from Android 4.0 (API 14), use an ActionProvider (the actual "SHARE")
  - Once attached to a menu item, handles both appearance and behavior

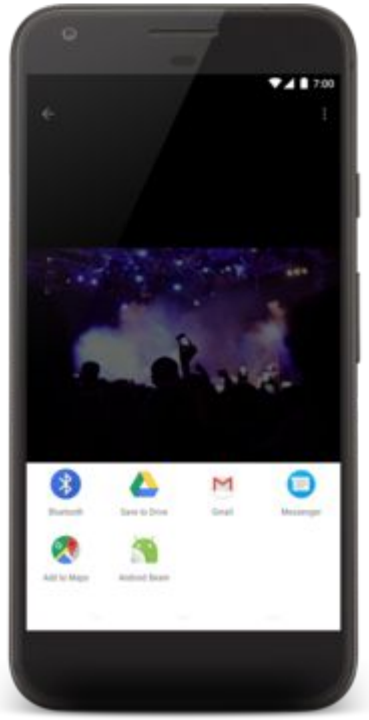
```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <item
         android:id="@+id/menu_item_share"
         android:showAsAction="ifRoom"
         android:title="Share"
         android:actionProviderClass=
            "android.widget.ShareActionProvider" />
   ...
</menu>
```



Sharing data (even easier!)

- You also need the appropriate ShareIntent
  - Once attached to a menu item, handles both appearance and behavior

```
private ShareActionProvider mShareActionProvider;

public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.share_menu, menu);
    MenuItem item = menu.findItem(R.id.menu_item_share);

    mShareActionProvider = (ShareActionProvider) item.getActionProvider();
    return true;
}

private void setShareIntent(Intent shareIntent) {
        mShareActionProvider.setShareIntent(shareIntent);
}
```



Navigation

Android Jetpack has launched the Android Navigation framework https://developer.android.com/guide/navigation

- Much easier way to handle navigation through:
  - **NavHostFragment** (in practice you have 1 Activity with many fragments interleaving in the NHF as container).
  - **NavigationController** (the central brain)
  - A **Navigation Graph**

Remember: Navigation is sourced into a Nav host fragment: an empty container within which the navigation takes place. There may be an Activity change, although infrequent.

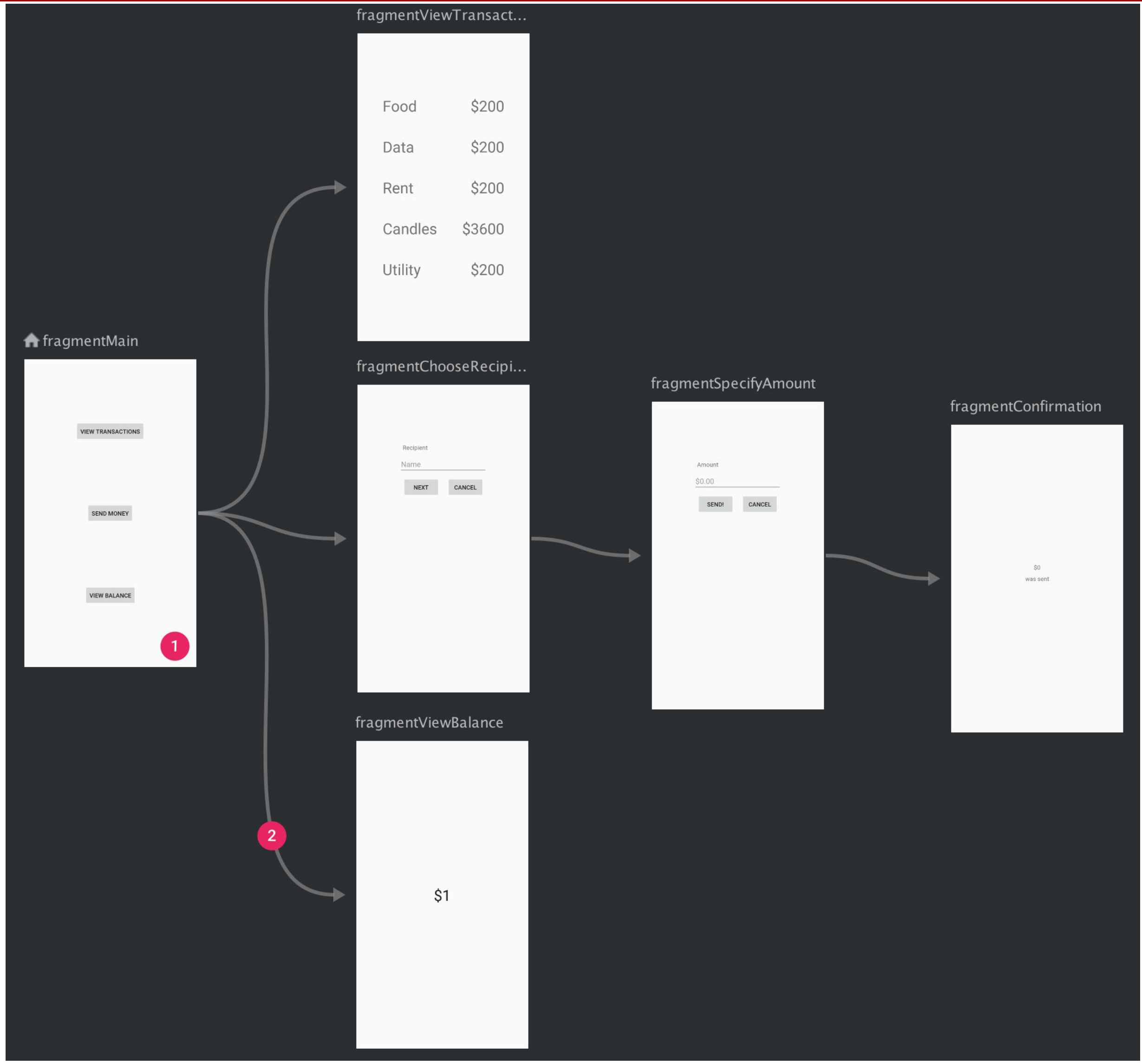
Navigation

Include the **Navigation** support:

```
dependencies {
  def nav_version = "2.3.5"
  // Java language implementation
  implementation "androidx.navigation:navigation-fragment:$nav_version"
  implementation "androidx.navigation:navigation-ui:$nav_version"

  // Kotlin
  implementation "androidx.navigation:navigation-fragment-ktx:$nav_version"
  implementation "androidx.navigation:navigation-ui-ktx:$nav_version"

  // Feature module Support
  implementation "androidx.navigation:navigation-dynamic-features-fragment:$nav_version"

  // Testing Navigation
  androidTestImplementation "androidx.navigation:navigation-testing:$nav_version"

  // Jetpack Compose Integration
  implementation "androidx.navigation:navigation-compose:1.0.0-alpha10"
}
```

The Navigation Graph:

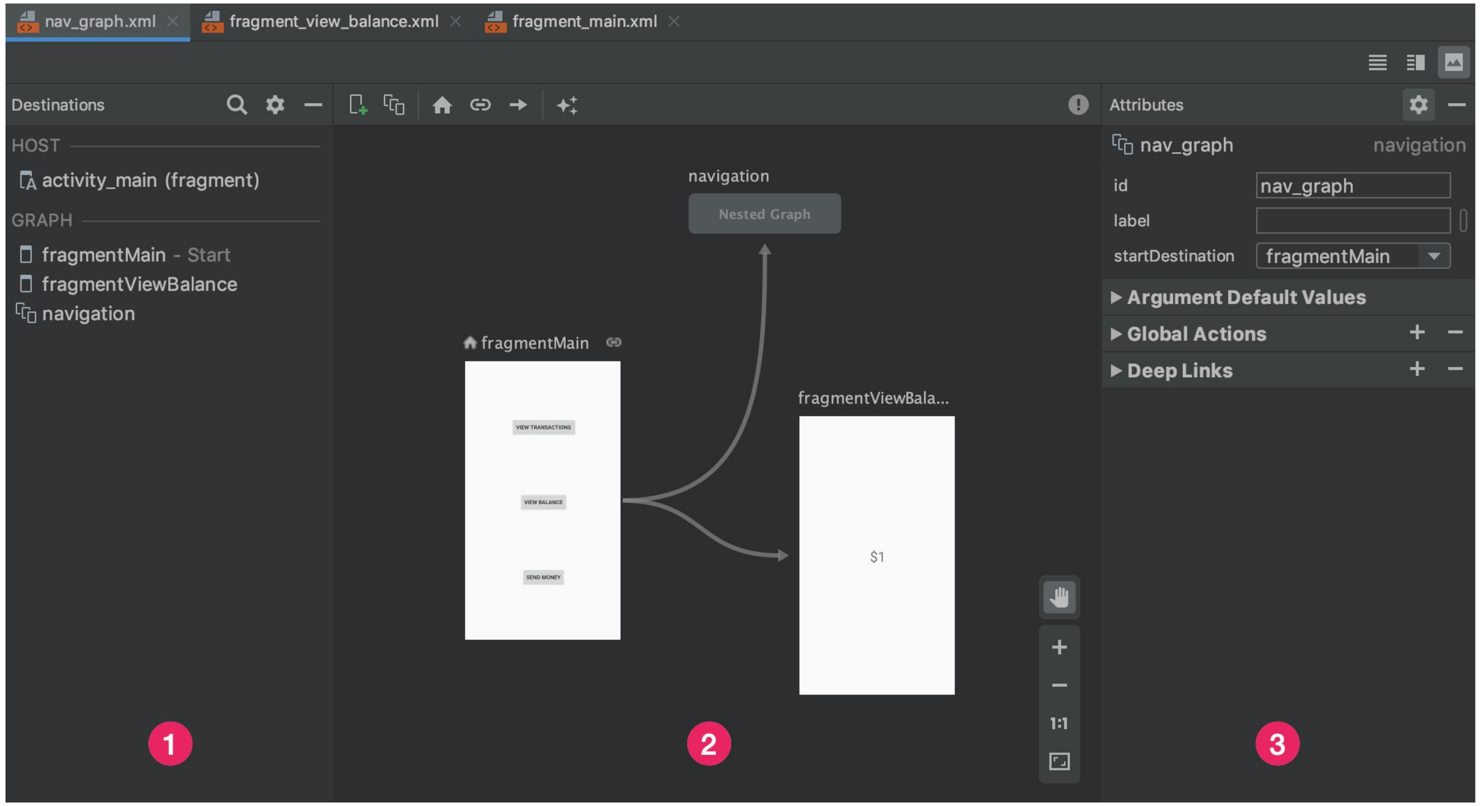
- An XML resource connecting **destinations** (fragments) through **actions** (events).
- The XML resource type is “navigation”
- It must take place within a NavHostFragment (although destinations can also be activities).



Food	$200
Data	$200
Rent	$200
Candles	$3600
Utility	$200

Navigation

You can edit the Navigation graph via the Navigation Editor.



1. Destination panel: you can see all your resources
2. Graph Editor: Contains a visual representation of your navigation graph. You can switch between Design view and the underlying XML representation in the Text view.
3. Attributes: Shows attributes for the currently-selected item in the navigation graph.

Navigation **Host**

Need to instantiate the **Nav Host** in the activity where you want the Navigation to take place. This is implemented automatically by a class called **NavHostFragment**

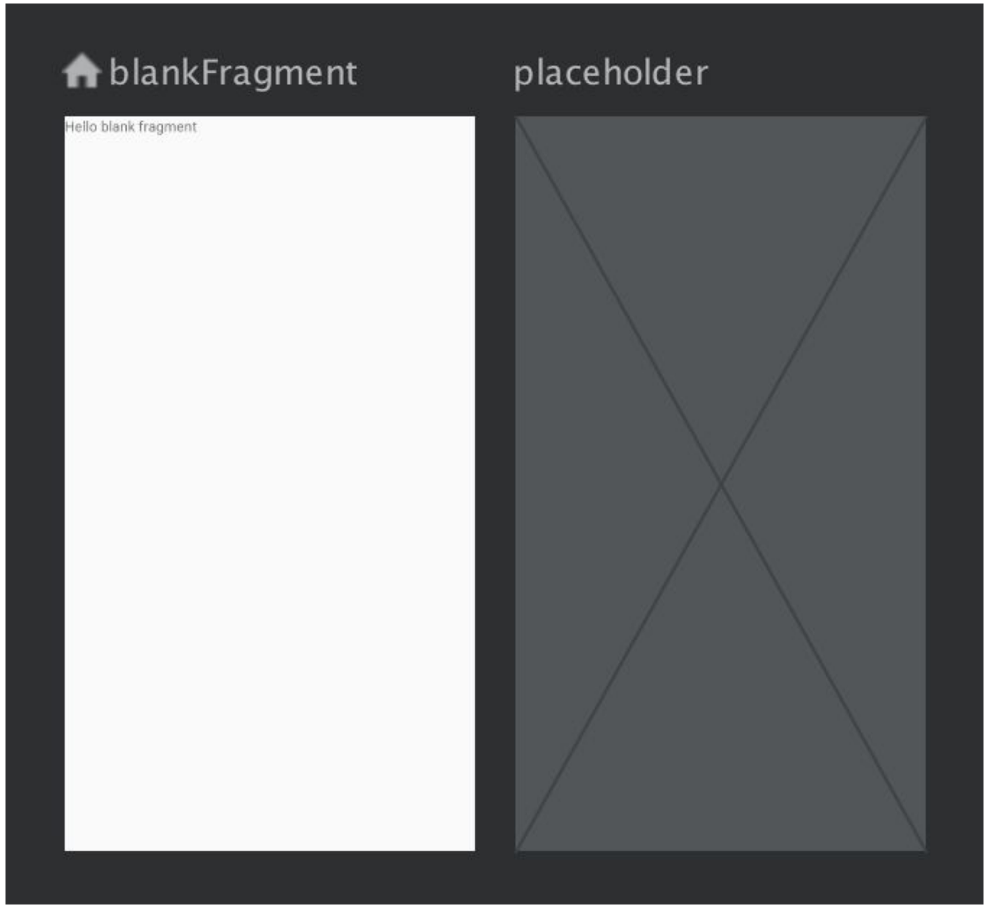
- Also specify to which navigation graph we are referring to by using the **navGraph** attribute.
- defaultNavHost allows the fragment to intercept the back button.

```
<androidx.fragment.app.FragmentContainerView
    android:id="@+id/nav_host_fragment"
    android:name="androidx.navigation.fragment.NavHostFragment"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintBottom_toBottomOf="parent"

    app:defaultNavHost="true"
    app:navGraph="@navigation/nav_graph" />
```

Create Destinations

In creating a destination through the Editor you need to specify 4 different fields:

- The **Type** field indicates whether the destination is implemented as a fragment, activity, or other custom class in your source code.
- The **Layout** field contains the name of the destination's XML layout file.
- The **ID** field contains the ID of the destination which is used to refer to the destination in code.
- The **Name** dropdown shows the name of the class that is associated with the destination. You can click this dropdown to change the associated class to another destination type.

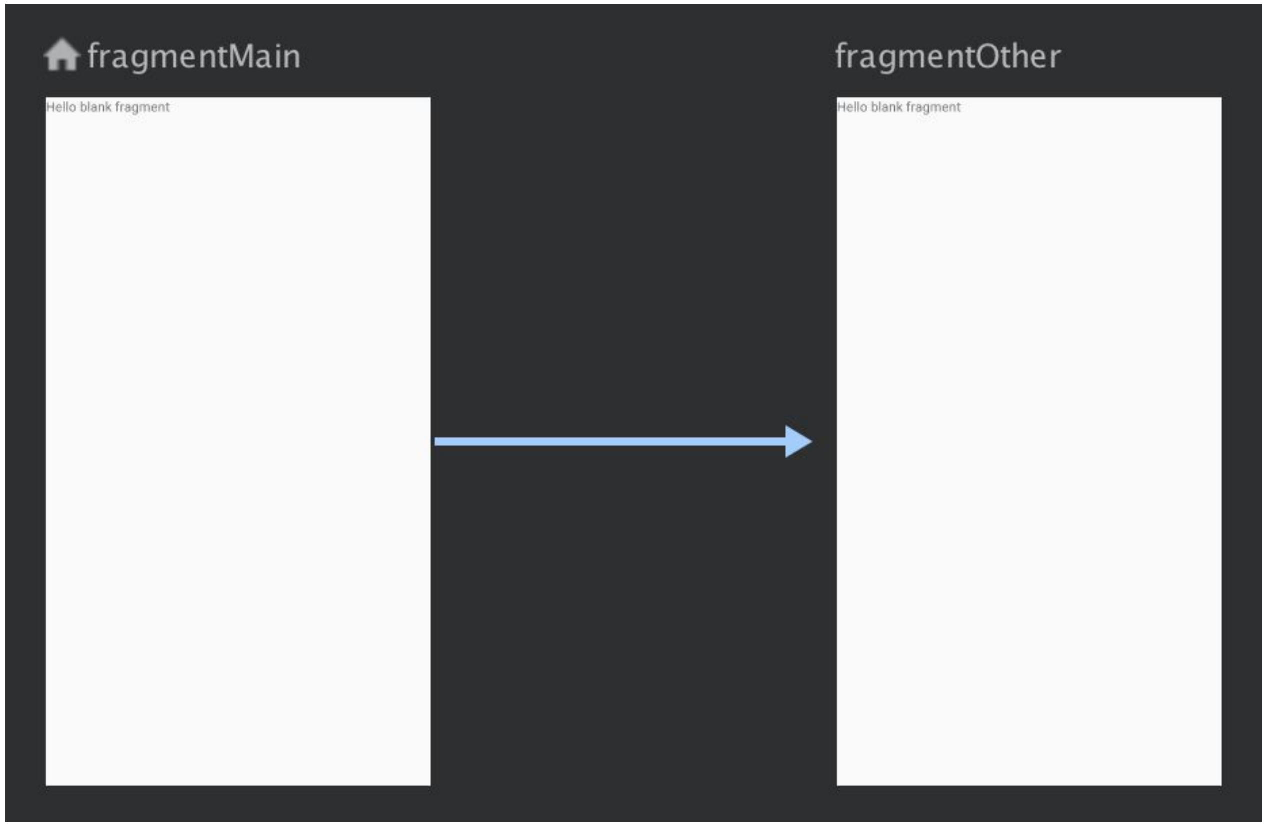


Create Actions

In creating an action through the Editor you need to connect two destinations and specify 3 different fields:

- The Type field contains “Action”.
- The ID field contains the ID for the action.
- The Destination field contains the ID for the destination fragment or activity.

```
<action
android:id="@+id/action_blankFragment_to
_blankFragment2"

app:destination="@id/blankFragment2"

/>
```



Navigation XML

Need to instantiate the **Nav Host** in the activity where you want the Navigation to take place. This is implemented automatically by a class called **NavHostFragment**

```
<navigation xmlns:app="http://schemas.android.com/apk/res-auto" xmlns:tools="http://schemas.android.com/tools"
xmlns:android="http://schemas.android.com/apk/res/android"
    app:startDestination="@id/blankFragment">
    <fragment
        android:id="@+id/blankFragment"
        android:name="com.example.cashdog.cashdog.BlankFragment"
        android:label="fragment_blank"
        tools:layout="@layout/fragment_blank" >
        <action
            android:id="@+id/action_blankFragment_to_blankFragment2"
            app:destination="@id/blankFragment2" />
    </fragment>
    <fragment
        android:id="@+id/blankFragment2"
        android:name="com.example.cashdog.cashdog.BlankFragment2"
        android:label="fragment_blank_fragment2"
        tools:layout="@layout/fragment_blank_fragment2" />
</navigation>
```

Navigation Controller

In order to perform an action we need to retrieve the NavHostFragment and obtain a reference to the NavController...

```
NavHostFragment navHostFragment =
        (NavHostFragment) supportFragmentManager.findFragmentById(R.id.nav_host_fragment);
NavController navController = navHostFragment.getNavController();
```

… and then simply navigate by declaring the action:

```
navController.navigate(R.id.action_blankFragment_to_blankFragment2);
```

Navigation keeps a backstack of all the transactions and overrides the usage of the back button to navigate back the backstack.

- It also sets a up button on the toolbar that does exactly the same thing as back, but it never exits the app (it is replaced by e.g. the navigation icon).
- It creates a fake backstack if we deep link to a certain screen.

SafeArgs

With SafeArgs we ensure type safety.
Add it to your classpath...

```
classpath "androidx.navigation:navigation-safe-args-gradle-plugin:2.3.5"
```

… and add the plugin to your module **build.gradle**

```
apply plugin: "androidx.navigation.safeargs"
```

- Once enabled, it creates a class for each origin destination ensuring type safety when performing an action. The class is called {*name_of_origin*} + “Directions”
- Such class has a method for each of the actions that returns a NavDirection object to be passed to the navigate function.

Considering the previous XML:

```
NavDirections action =
      BlankFragmentDirections
         .action_blankFragment_to_blankFragment2();
Navigation.findNavController(view).navigate(action);
```

Principles of Material Design

“We challenged ourselves to create a visual language for our users that synthesizes the classic principles of good design with the innovation and possibility of technology and science.“

Design which spans through different platforms (Android, iOS, Web, Flutter)

3 main principles:

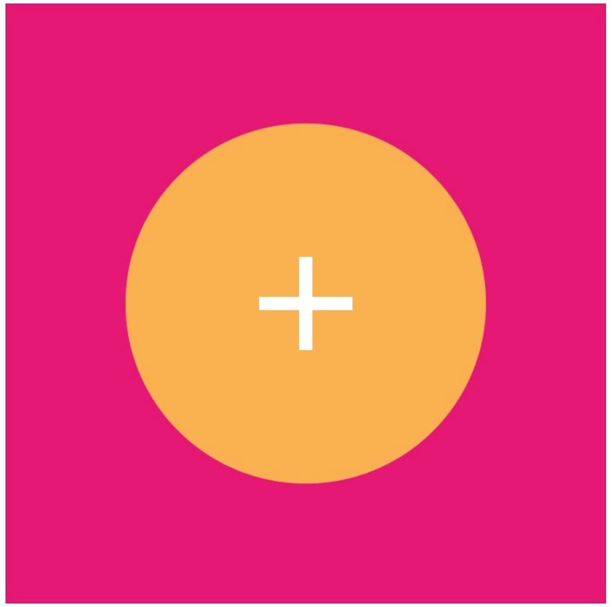
- Material is the metaphor
- Bold, graphic, intentional
- Motion provides meaning

Material is the metaphor

- Rationalized space and system of motion
- Inspired by paper and ink, but technologically advanced
- Surface and edges should provide visual cues
  - Stick to physic rules
- Light, surfaces, shadows

Bold, graphic, intentional

- To create a hierarchy
- Emphasis on user actions
- Color heavily matter
  - Use of edge-to-edge decorations and specialized typography is key
- Not just to please the eye



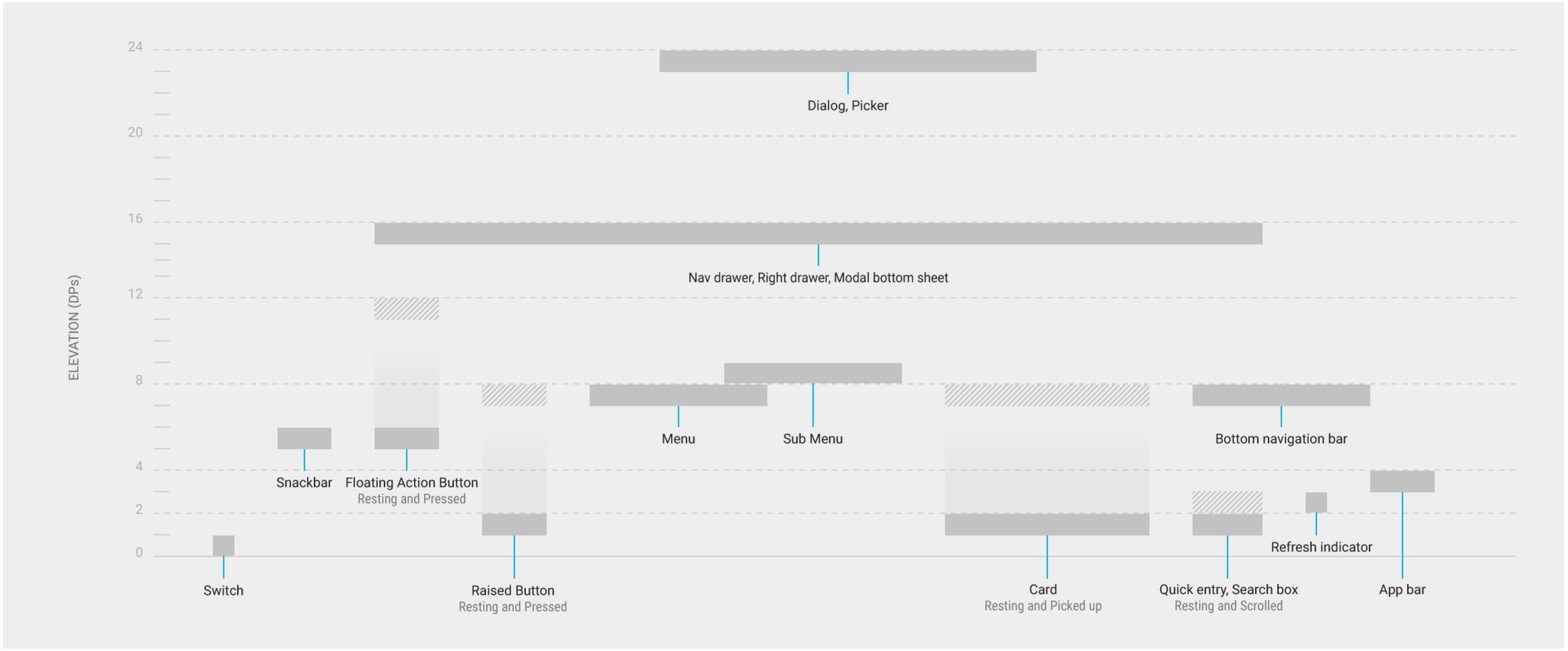
Motion provides meaning

- Bound to user actions
- User movements initiate a change in the layout
- Should not break the design continuity, even though objects are moving
- Motion is meaningful:
  - Not just to animate, but to provide feedback

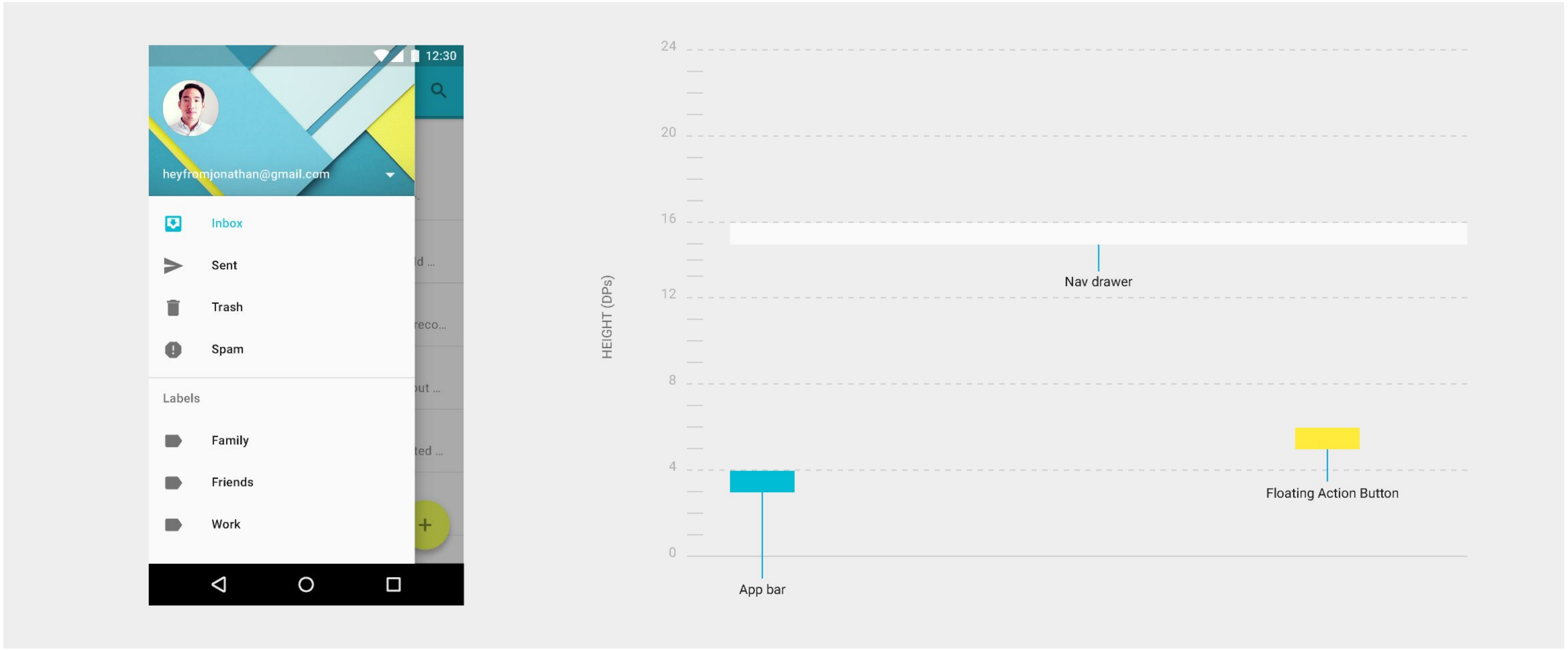
MaterialDesign Environment

- MaterialDesign is a 3D environment
  - Each object has x,y,z values and thickness (1dp)
- Each object is on a different layer, providing elevation and shadowing lower layers
- Each object material is solid
  - Events cannot span through different materials
  - Multiple materials cannot occupy the same point in space
- Materials can change places and shape
  - But do not fold or bend

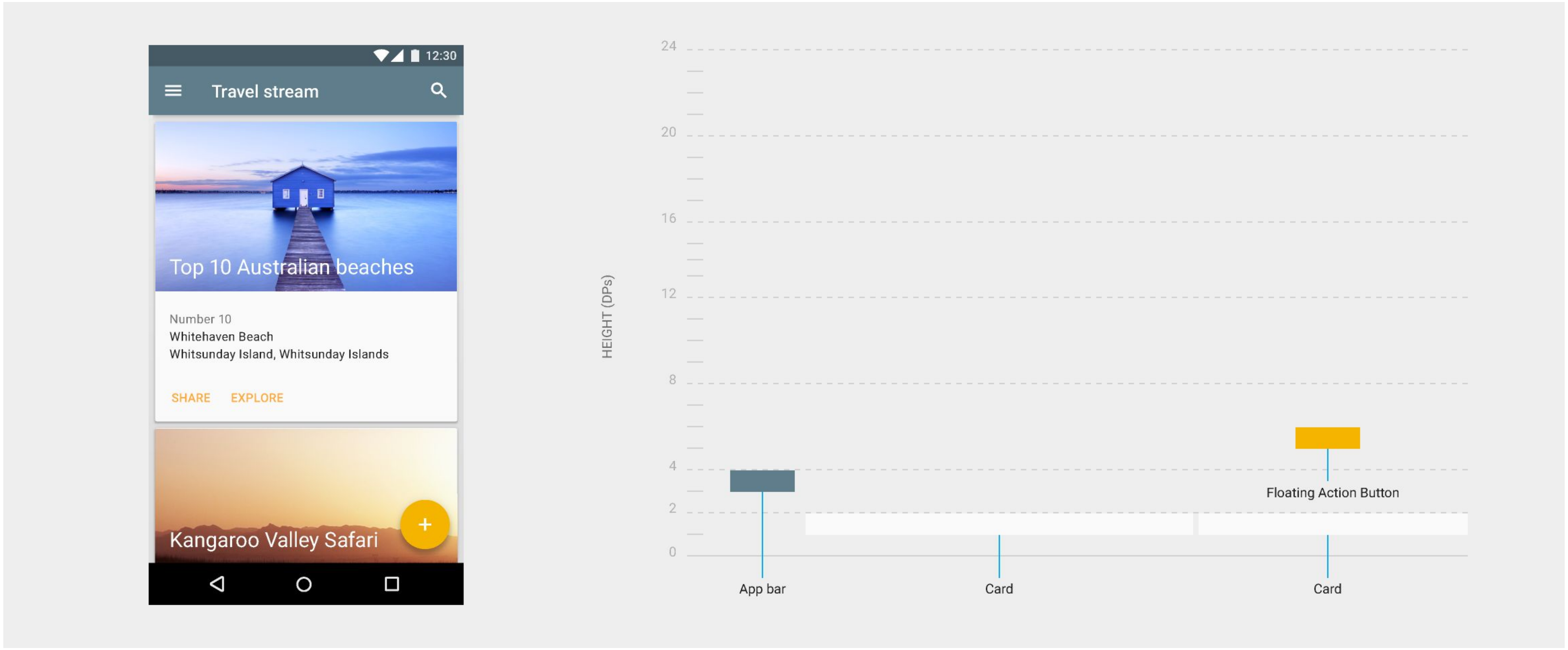
Elevation example



Elevation example

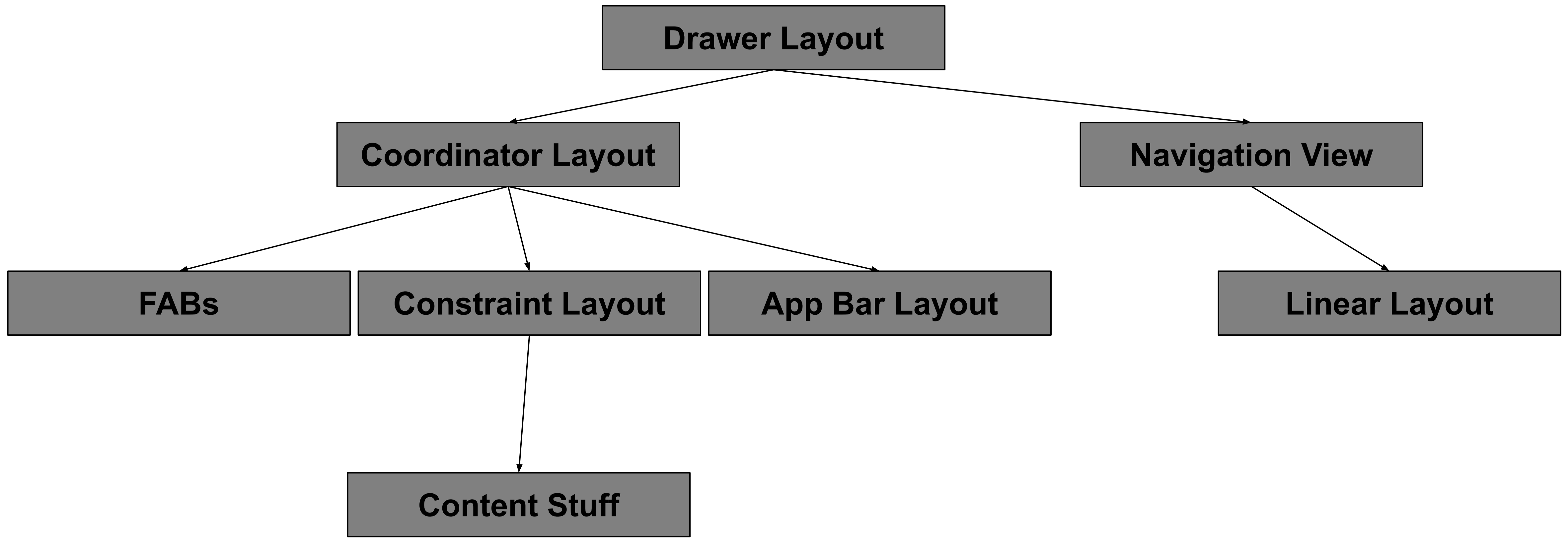


Elevation example



Layout Example

Often Layout organization reflects elevation...



Drawer Layout
Coordinator Layout
Navigation View
FABs
Constraint Layout
App Bar Layout
Linear Layout
Content Stuff