EMBEDDED SYSTEMS PROGRAMMING 2014-15

UI and Android

ANDROID: STANDARD GESTURES (1/2)

UI classes inheriting from `View` allow to set **listeners** that respond to basic gestures. Listeners are defined by suitable **interfaces**.

- **`boolean onTouch(View v, MotionEvent event)`**
  Part of the `View.OnTouchListener` interface.
  The user has performed an action qualified as a touch event, including a press, a release, or any movement gesture on the screen (within the bounds of the item).

- **`void onClick(View v)`**
  Part of the `View.OnClickListener` interface.
  The user has touched the item

- **`boolean onLongClick(View v)`**
  Part of the `View.OnLongClickListener` interface.
  The user has touched and holds the item

EXAMPLE

```
public class ExampleActivity extends Activity implements OnTouchListener
{
    protected void onCreate(Bundle savedValues)
    {
...
        Button button = (Button)findViewById(R.id.bu);
        button.setOnTouchListener(this);
    }

    // Implement the OnTouchListener callback
    public boolean onTouch(View v, MotionEvent event)
    {
      // do something when the button is clicked

      return true;
    }
...
}
```

ANDROID:
STANDARD GESTURES (2/2)

- Standard UI widgets respond to standard gestures (e.g., a `ListView` responds to a flick)
- Custom UI widgets can handle touch screen motion events by implementing the **`onTouchEvent(MotionEvent event)`** method; no gesture recognizer is provided

SUPPORTING DIFFERENT SCREENS (1/3)

- Mobile platforms support a variety of devices with different screen sizes and resolutions
- Resolution does not cont that much: **it is size that matters**
  - **Bigger screens** can accommodate more information than smaller screens
  - **Tablet screens** can accommodate more information than other screens

SUPPORTING DIFFERENT SCREENS (2/3)

- Different screen sizes may require different artwork
- Different screen sizes typically require different UIs
  - Use more / resize conventional UI elements
  - Introduce **new UI elements that are specifically designed for tablets**

SCREENS: ANDROID

- Tens of locales (e.g., `-en-rUS`), device dependent
- Four generalized screen sizes:
  small (`-small`), normal (`-normal`), large (`-large`),
  extra large (`-xlarge`)
- Two variations of each screen size:
  portrait (`-port`), landscape (`-land`)
- Four generalized screen densities:
  120 DPI (`-ldpi`), 160 DPI (`-mdpi`), 240 DPI (`-hdpi`),
  320 DPI (`-xhdpi`), 480 DPI (`-xxhdpi`), 640 DPI (`-xxxhdpi`)
- Place custom resources in the appropriate folder: Android will use them

SCREENS: EXAMPLES

- Directory for default layouts: "`res/layout`"
- Directory for layouts that target large screens and the portrait orientation: "`res/layout-large-land`"
- Directory for default artwork: "`res/drawable`"
- Directory for artwork that target US-English devices in landscape orientation: "`res/drawable-en-rUS-land`"

- For a full list of directories and modifiers, look up the "Providing resources" page in the Android documentation
- **Always provide default resources** (i.e., a folder with no modifiers)

TABLETS: ANDROID

- Up to version 2.3 (API level ≤ 10):
  no support for tablets

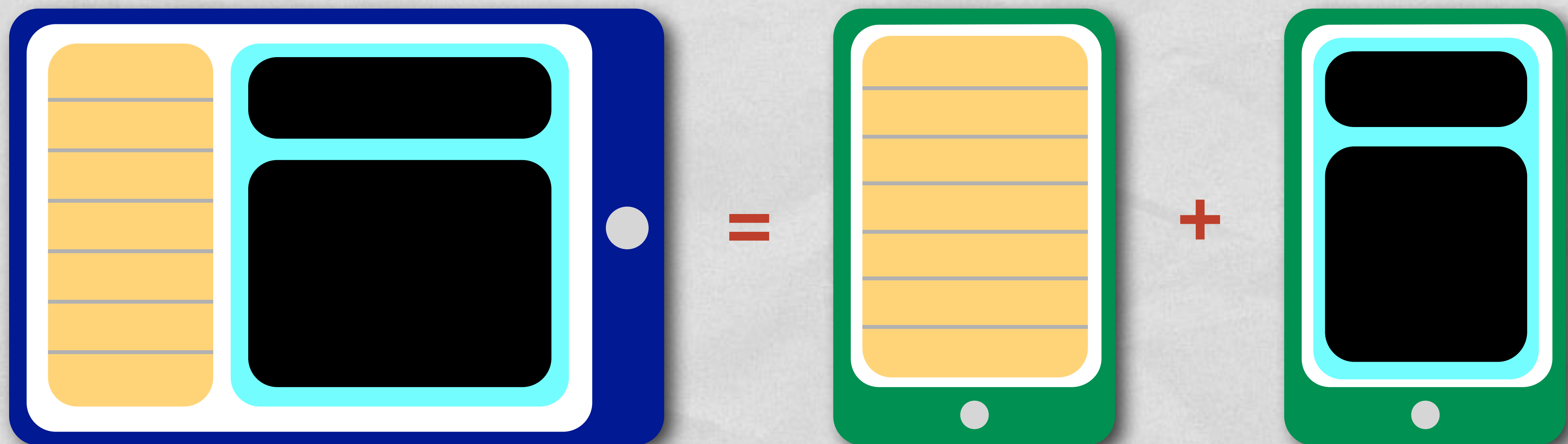
- 3.x versions (11 ≤ API level ≤ 13):
  run only on tablets

- Version 4.0 and above (API level ≥ 14):
  unified support for tablets and other devices

MULTI-PANE LAYOUTS

- From `developer.android.com`:

> *The most effective way to create a distinct user experience for tablets and handsets is to [...]* ***design "multi-pane" layouts for tablets and "single-pane" layouts for handsets.***



SUPPORTING DIFFERENT SCREENS (3/3)

- Implement flexible layouts and provide multiple version of relevant resources
- Design activities using **fragments**
- Use the **action bar**

FRAGMENT CLASS

- Introduced in Android 3.0 (API level 11)
- Represents a portion of user interface
- **Hosted by an activity**: to be precise, it "lives" in a `ViewGroup` inside the activity's view hierarchy, albeit it **defines its own view layout** and **has its own lifecycle callbacks**
- Each fragment can be manipulated independently from other fragments

FRAGMENT: LIFECYCLE

- A class derived from `Fragment` behaves similarly to an activity. It includes lifecycle callback methods (`onCreate()`, etc.)
- Two additional methods: **`onCreateView()`** and **`onDestroyView()`**
- Lifecycle callback methods **must be invoked by the hosting activity**

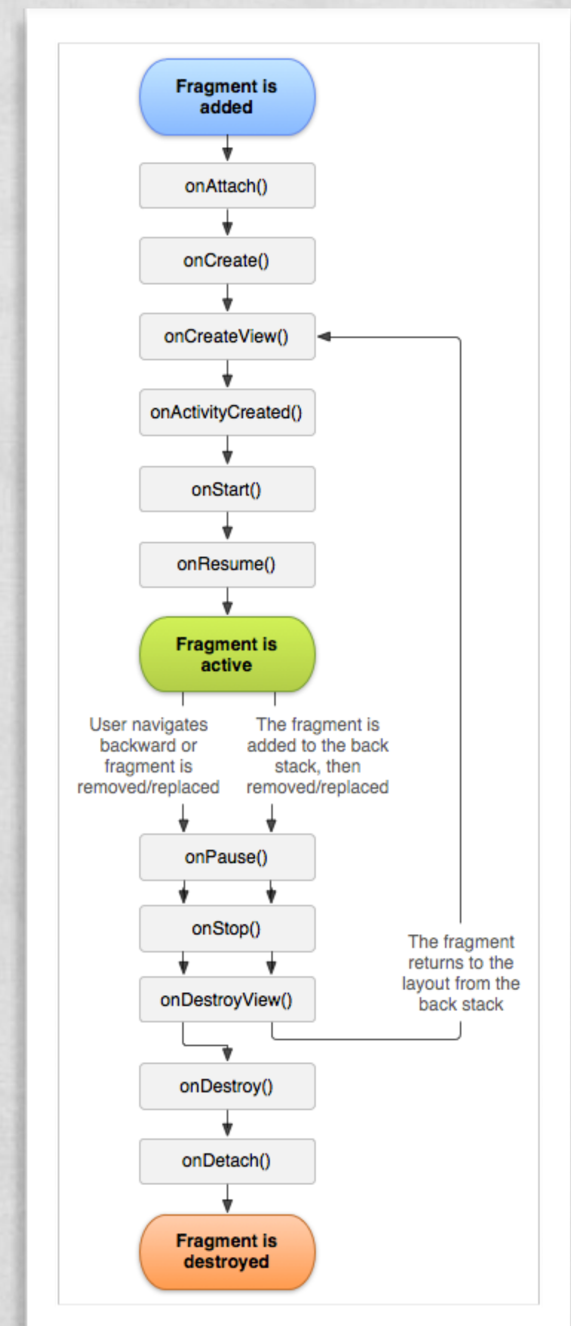


Image from Android Developer

HOSTING A FRAGMENT

- **Declarative approach**: add the fragment to the layout file of the hosting activity
- **Programmatic approach**: add the fragment in the source code of the hosting activity; instantiate the UI in the `onCreateView()` method of the fragment

ONCREATEVIEW, ONDESTROYVIEW METHODS

- **`View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)`**
  Instantiates the UI for a fragment and attaches it to `container`

- An implementation for `onCreateView()` must be provided by the programmer

- If the UI is defined in an XML file, the system-provided `LayoutInflater` can be used to instantiate (“inflate”) it

- **`void onDestroyView()`**
  Destroys a previously-created user interface

FRAGMENTMANAGER: TWO KEY METHODS

- An instance of the `FragmentManager` class allows interaction with fragments. For instance, it allows to add or remove a fragment (via a **fragment transaction**)
- **`Fragment findFragmentById(int id)`**
  Returns the fragment which is identified by the given `id` (as specified, e.g., in the XML layout file)
- **`FragmentTransaction beginTransaction()`**
  Start editing the Fragments associated with the `FragmentManager`. The transaction is ended by invoking the `commit()` method of `FragmentTransaction`

HOSTING A FRAGMENT: EXAMPLE (1/2)

- **Declarative approach**
  2 fragments declared inside the layout of an activity.
  When the activity is created, instances of the classes associated with the fragments are automatically allocated

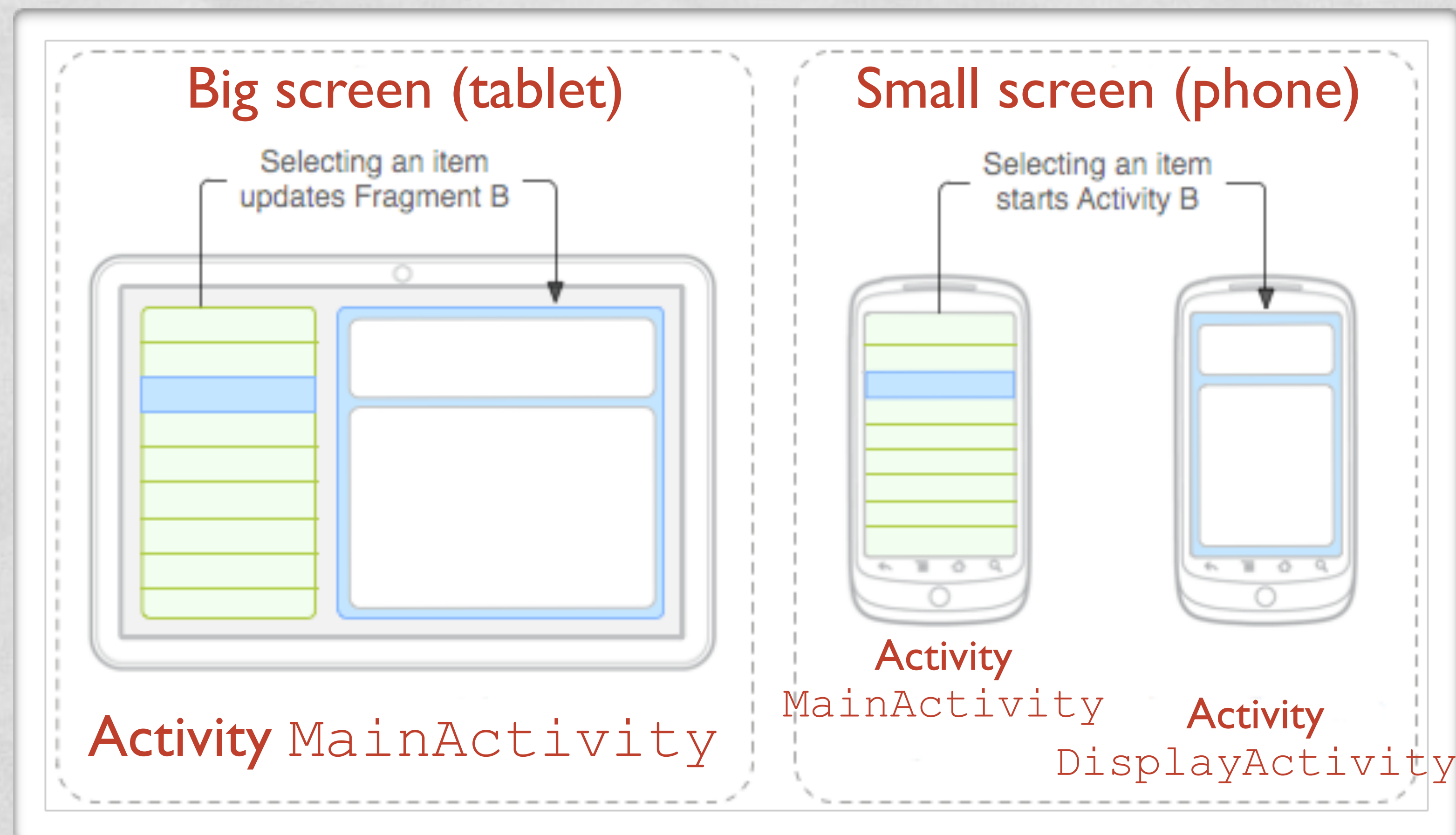
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment android:name="it.unipd.dei.esp1112.email.ListFragment"
            android:id="@+id/list"
            android:layout_weight="1"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
    <fragment android:name="it.unipd.dei.esp1112.email.ReaderFragment"
            android:id="@+id/reader"
            android:layout_weight="2"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
</LinearLayout>
```

HOSTING A FRAGMENT: EXAMPLE (2/2)

- **Programmatic approach**
  Initiate a fragment transaction, instantiate a fragment, then add it to a suitable `ViewGroup`

```
...

FragmentManager fragmentManager = getFragmentManager()
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();

ExampleFragment fragment = new ListFragment();

fragmentTransaction.add(R.id.fragment_container, fragment);
fragmentTransaction.commit();

...
```

FRAGMENTS: EXAMPLE (1/4)



Example from Android Developers

- **DisplayActivity** is started only if the screen is small.
  It hosts fragment `DetailsFragment`
- **MainActivity** always manages fragment `TitlesFragment` and, depending on the screen size, hosts `DetailsFragment` as well or starts `DisplayActivity`

FRAGMENTS: EXAMPLE (2/4)

- Layout for `MainActivity`, big screen
- Resides in `res/layout-large/`
- Both fragments are hosted by `MainActivity`

```
<?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"
  android:id="@+id/frags">
    <!-- "Fragment A" -->
  <fragment class="com.example.android.TitlesFragment"
            android:id="@+id/list_frag"
            android:layout_width="@dimen/titles_size"
            android:layout_height="match_parent"/>
    <!-- "Fragment B" -->
  <fragment class="com.example.android.DetailsFragment"
            android:id="@+id/details_frag"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />
</LinearLayout>
```

FRAGMENTS: EXAMPLE (3/4)

- Layout for `MainActivity`, small screen
- Resides in `res/layout/`
- `DisplayFragment` is hosted by `DisplayActivity`

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <!-- "Fragment A" -->
    <fragment class="com.example.android.TitlesFragment"
              android:id="@+id/list_frag"
              android:layout_width="match_parent"
              android:layout_height="match_parent"/>
</FrameLayout>
```

FRAGMENTS: EXAMPLE (4/4)

- Code snippet from the `MainActivity` class

```
public class MainActivity extends Activity implements TitlesFragment.OnItemSelectedListener
{
    ...

    /** TitlesFragment.OnItemSelectedListener is a callback that the list fragment
        ("DetailsFragment") calls when a list item is selected */
    public void onItemSelected(int position)
    {
        DisplayFragment displayFrag = (DisplayFragment) getFragmentManager()
                                   .findFragmentById(R.id.display_frag);
        if (displayFrag == null)
        {
            // DisplayFragment is not in the layout (handset layout),
            // so start DisplayActivity (Activity B)
            // and pass it the info about the selected item
            Intent intent = new Intent(this, DisplayActivity.class);
            intent.putExtra("position", position);
            startActivity(intent);
        }
        else
        {
            // DisplayFragment is in the layout (big screen layout),
            // so tell the fragment to update
            displayFrag.updateContent(position);
        }
    }
}
```

ACTION BAR (1/2)

Picture: Android Design

UI component that can contain, from left to right,

1. the application **icon**,
2. the **view control** (tabs or a spinner),
3. a certain number of **action items**,
4. the **action overflow menu button**

May also contain a hint to the **navigation drawer**

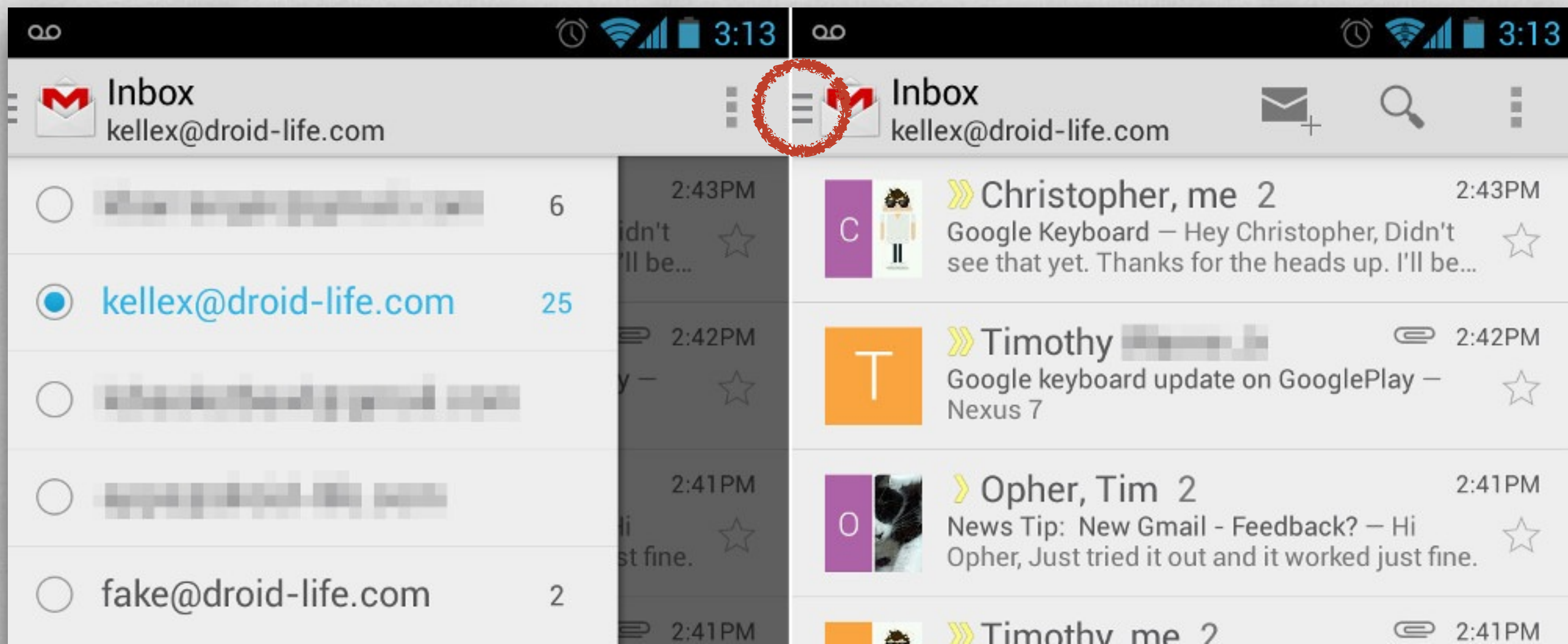
APP BAR

- Android 5.x alternatively uses the name "app bar"
- The nav icon, if present can be:
    - an arrow for navigating the app's hierarchy
    - a control to open a navigation drawer

Picture: Android Design

NAVIGATION DRAWER

- Displays the main navigation options for the app
- Appears from the left side of the screen by clicking on the application icon



OVERFLOW MENU

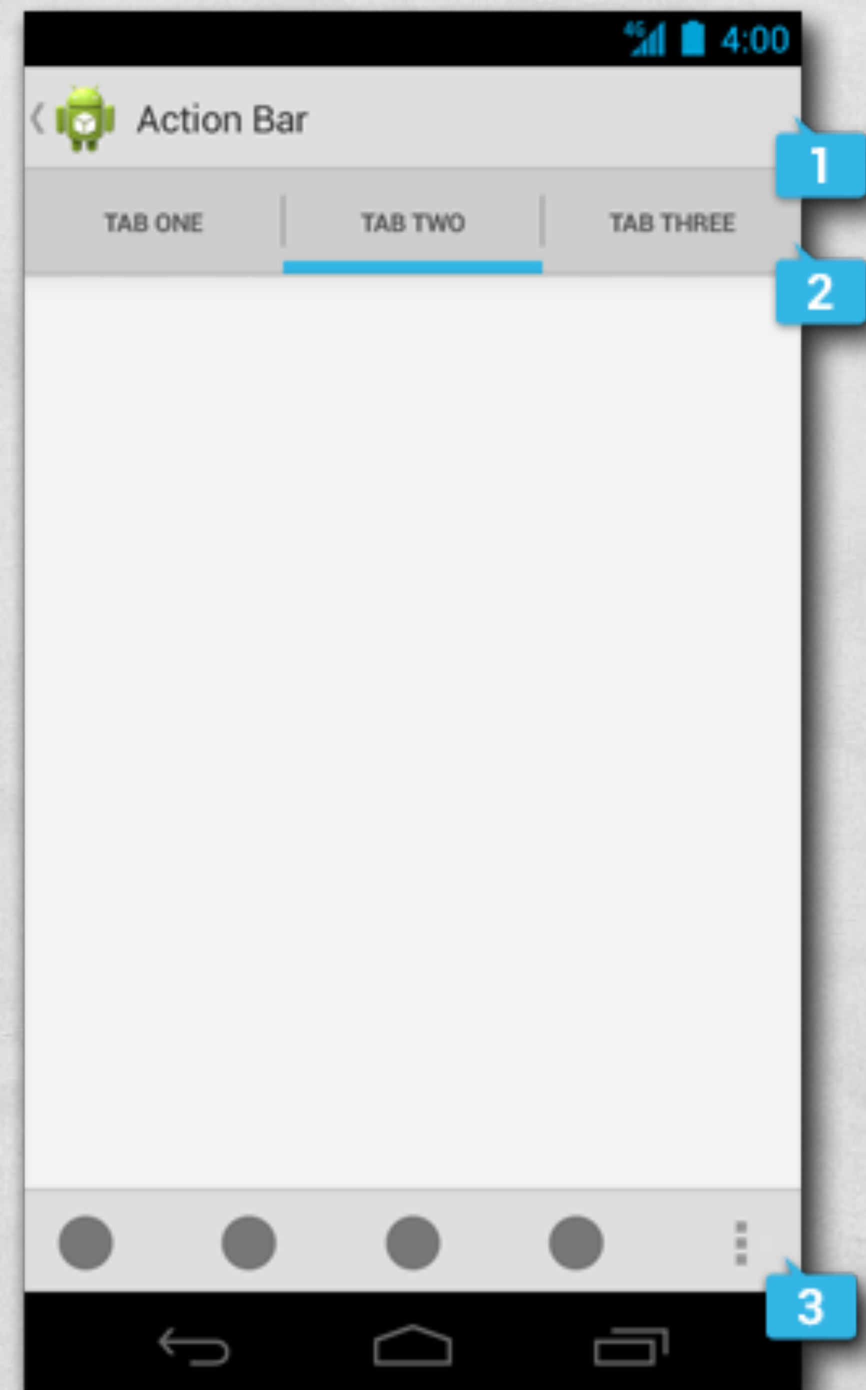
- Groups action items that are not important enough to be prominently displayed in the action bar
- Duplicates the functionality of the option menu + the (hardware) menu button
- In Android 4.0+, developers are strongly encouraged to migrate to the overflow menu

Image from Android Design

ACTION BAR (2/2)

Depending on the screen size, content may be split across multiple action bars:

1. main action bar,
2. top bar,
3. bottom bar



Picture: Android Design

ADDING THE ACTION BAR

- Beginning with Android 3.0, an action bar is **created by default** for every application that declares a `targetSdkVersion` of 11 or greater in its manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package= ... >

<uses-sdk android:minSdkVersion="8"
          android:targetSdkVersion="11" />

    <application
        ...
    </application>

</manifest>
```

ACTION BAR: ADDING ITEMS

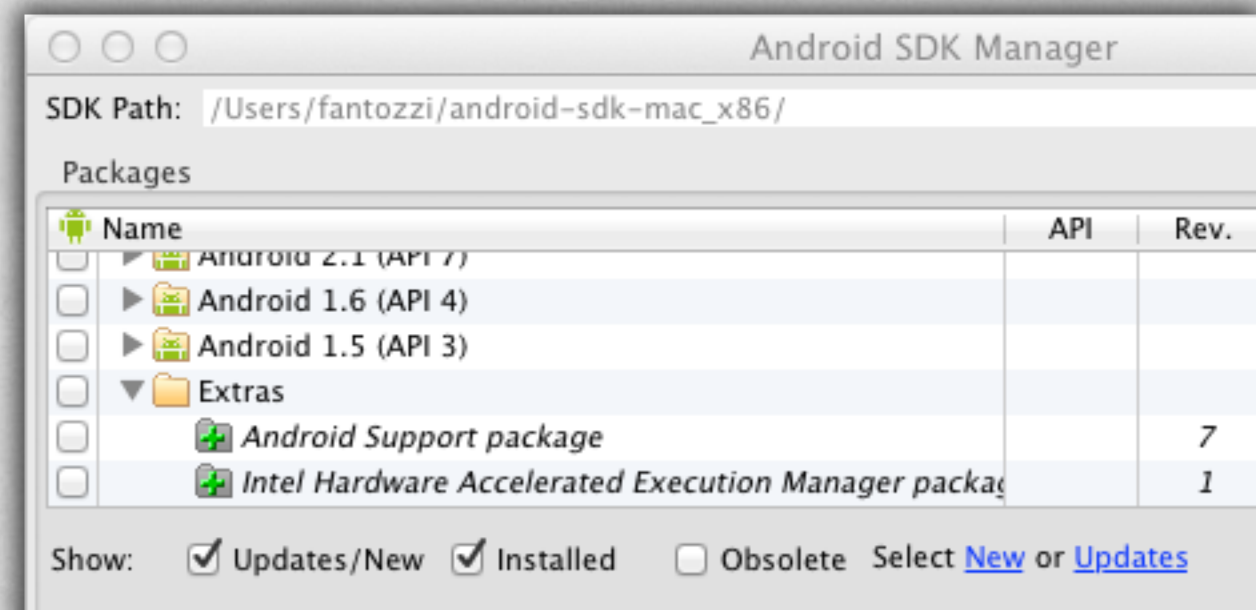
- The action bar can be populated in the **`onCreateOptionsMenu()`** activity method, which is called when the activity starts
- Action items and overflow menu items are managed together as a **menu resource**.
  The **`onOptionsItemSelected()`** activity method is called whenever an item is selected by the user
- If the action bar is constrained for space, some action items can be moved to the overflow menu

SUPPORT PACKAGE (1/2)

- **Android Support Package**: provides static libraries that can be added to an Android application in order to use APIs that are either not available on older platform versions or not part of the framework APIs.

- Each library runs only on devices that provide at least a **minimum API level**

- Must be installed from the Android SDK Manager



SUPPORT PACKAGE (2/2)

- **v4 Support Library**
  - Minimum API level: 4 (Android 1.6+)
  - Provides support for fragments and navigation drawers
- **v7 Appcompat Library**
  - Minimum API level: 7 (Android 2.1+)
  - Provides support for action bars
- More libraries available

SUPPORT LIBRARY: FRAGMENTS

- **`android.support.v4.app.Fragment`**, **`android.support.v4.app.FragmentActivity`** and **`android.support.v4.app.FragmentManager`** classes, to name a few, re-implement fragment support

- Use such classes to write a single piece of code that runs on any API level ≥ 4
- Host your fragments inside a `FragmentActivity`
- To get the `FragmentManager`, invoke `getSupportFragmentManager()`

REFERENCES

- Android User Interface
- Supporting Different Screens
- Supporting Tablets and Handsets
- Building a Dynamic UI with Fragments
- Designing for Multiple Screens
- Designing for Seamlessness

LAST MODIFIED: MAY 14, 2015

COPYRIGHT HOLDER: CARLO FANTOZZI (FANTOZZI@DEI.UNIPD.IT)
LICENSE: CREATIVE COMMONS ATTRIBUTION SHARE-ALIKE 3.0