Multiple devices

Three elements

- Use view dimensions that allow the layout to resize
- Create alternative UI layouts according to the screen configuration
- Provide bitmaps that can stretch with the views

Multiple devices

- Use wrap_content and match_parent
- Use RelativeLayout/ConstraintLayout
  - With ConstraintLayout, we should not use match_parent. Instead, set the dimension to 0dp to enable a special behavior called "match constraints»
- Define configuration-specific alternatives for a set of resources

Configuration-specific resources

Screen size	**small** **normal** **large** **xlarge**

- Create a new directory in res/ named in the form *<resources_name>-<config_qualifier>*
  - *<resources_name>* is the directory name of the corresponding default resources
  - *<qualifier>* is a name that specifies an individual configuration for which these resources are to be used

smallestWidth	**sw<N>dp** Examples: **sw320dp** **sw600dp** **sw720dp**	The fundamental size of a screen, as indicated by the shortest dimension of the available screen area. Specifically, the device's smallestWidth is the shortest of the screen's available height and width (you may also think of it as the "smallest possible width" for the screen). You can use this qualifier to ensure that, regardless of the screen's current orientation, your app's has at least **<N>** dps of width available for its UI.

Examples (smallestWidth)

- 320, for
  - 240x320 ldpi (QVGA handset)
  - 320x480 mdpi (handset)
  - 480x800 hdpi (high-density handset)
- 480 for 480x800 mdpi (tablet/handset)
- 600, for 600x1024 mdpi (7" tablet).
- 720, for 720x1280 mdpi (10" tablet).

Examples

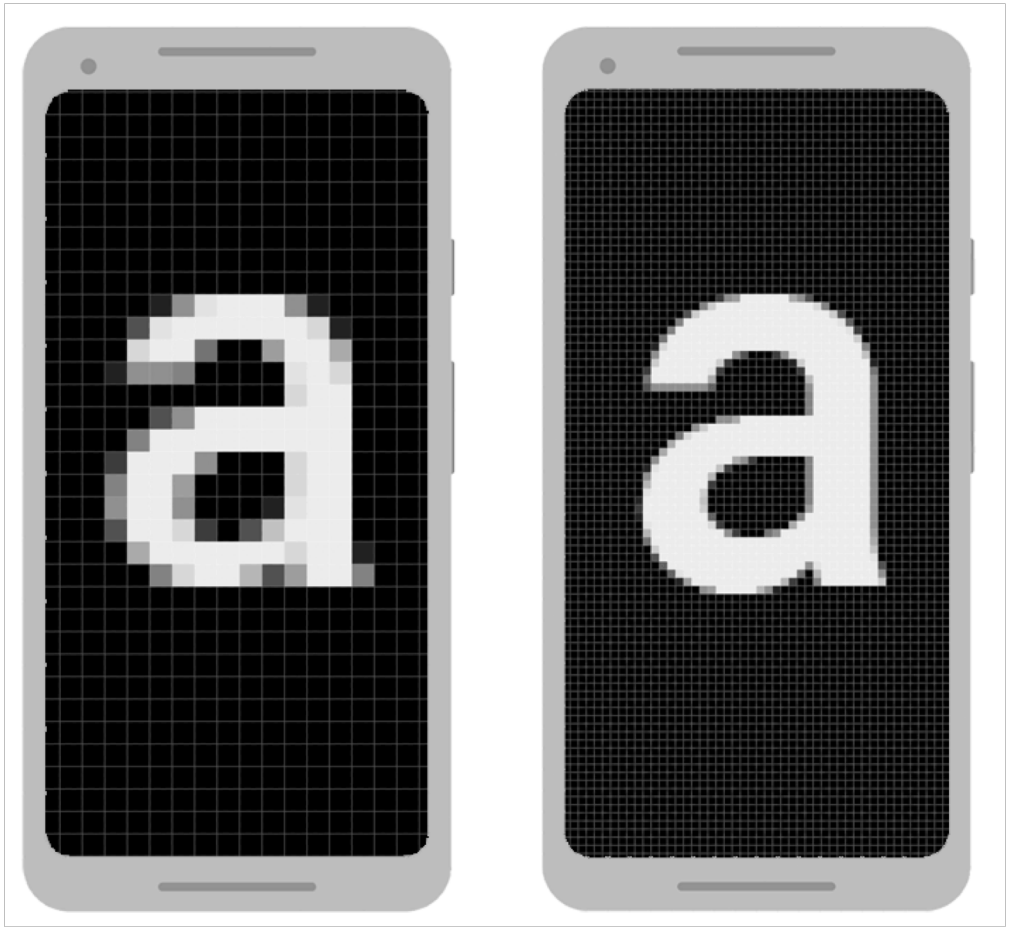
```
res/layout/main_activity.xml          # For handsets (smaller than 600dp available width)
res/layout-w600dp/main_activity.xml   # For 7” tablets or any screen with 600dp
                                      #   available width (possibly landscape handsets)
```

```
res/layout/main_activity.xml                # For handsets
res/layout-land/main_activity.xml           # For handsets in landscape
res/layout-sw600dp/main_activity.xml        # For 7” tablets
res/layout-sw600dp-land/main_activity.xml   # For 7” tablets in landscape
```

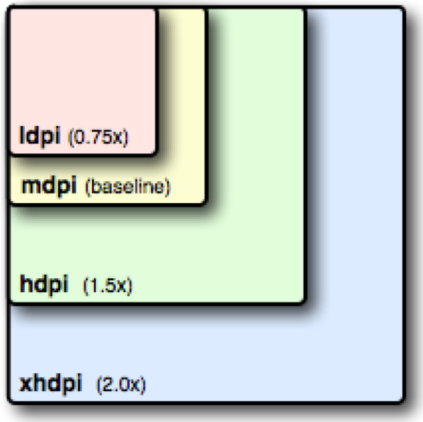
- Android 3.1 (API level 12)

```
res/layout/main_activity.xml            # For handsets (smaller than 640dp x 480dp)
res/layout-large/main_activity.xml      # For small tablets (640dp x 480dp and bigger)
res/layout-xlarge/main_activity.xml     # For large tablets (960dp x 720dp and bigger)
```

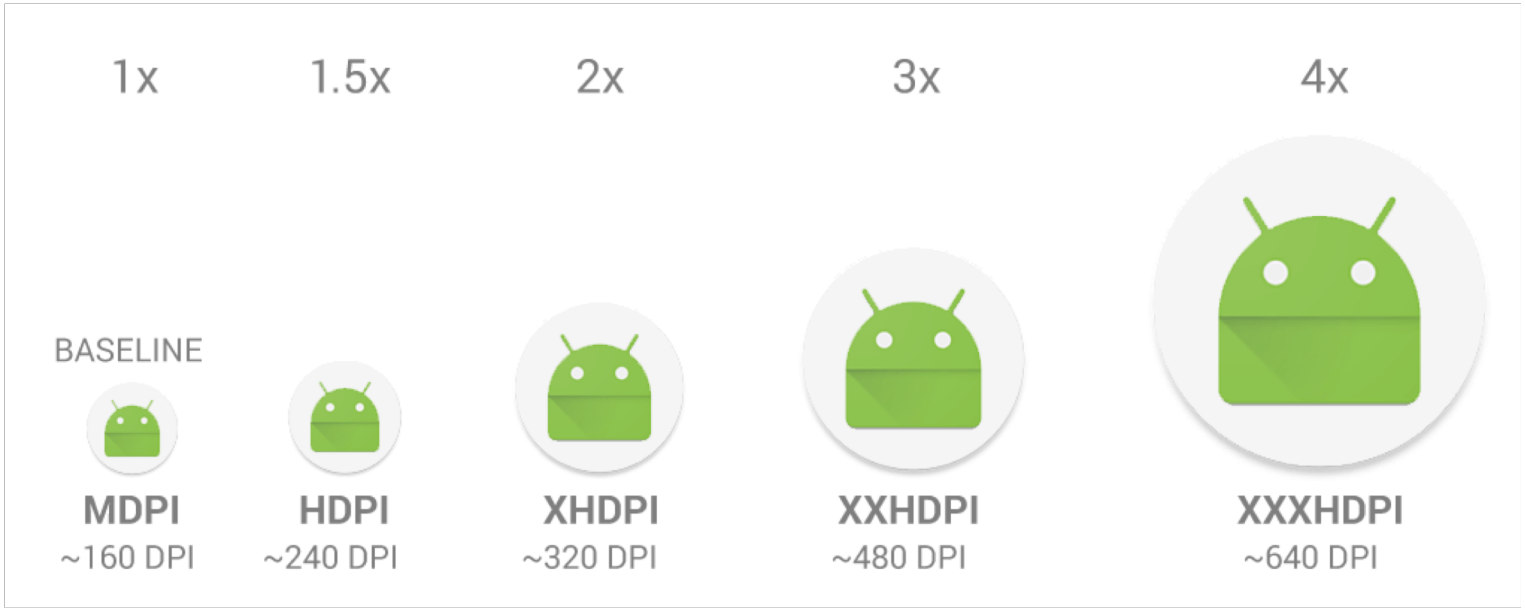
Screen density



Density



- px = dp * (dpi / 160)
- For example Android says that Launcher icons on a mobile device must be 48x48 dp



Bipmaps

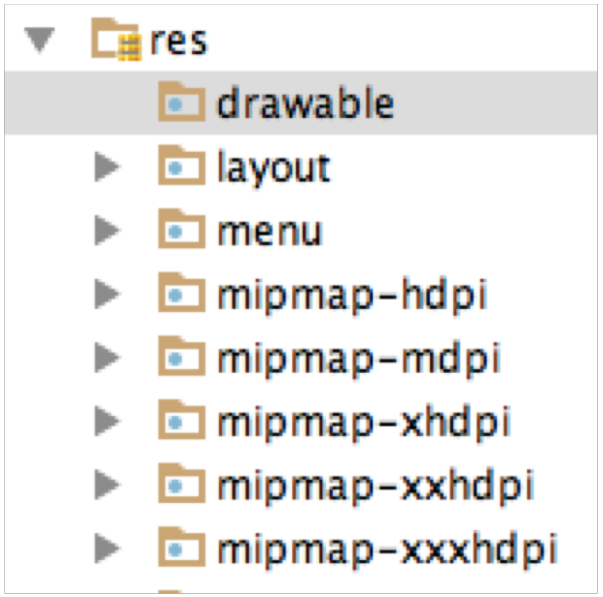
- The system uses any size- or density-specific resources from your application and displays them without scaling
  - If resources are not available in the correct density, the system loads the default resources and scales them up or down as needed
- The system assumes that default resources (those from a directory without configuration qualifiers) are designed for the baseline screen density (mdpi)
  - A bitmap designed at 50x50 pixels for an mdpi screen is scaled to 75x75 pixels on an hdpi screen (if there is no alternative resource for hdpi)

Manifest and devices

- We must also declare in the manifest file which screens our application supports
  - Through <supports-screens> manifest element
  - if your application supports all screen sizes supported by Android (as small as 426dp x 320dp), then you don't need to declare this attribute, because the smallest width your application requires is the smallest possible on any device

Guidelines

- Use wrap_content, match_parent, or the dp unit for layout dimensions
- Do not use hard-coded pixel values in your application code
- Do not use AbsoluteLayout (deprecated)
- Use size and density-specific resources



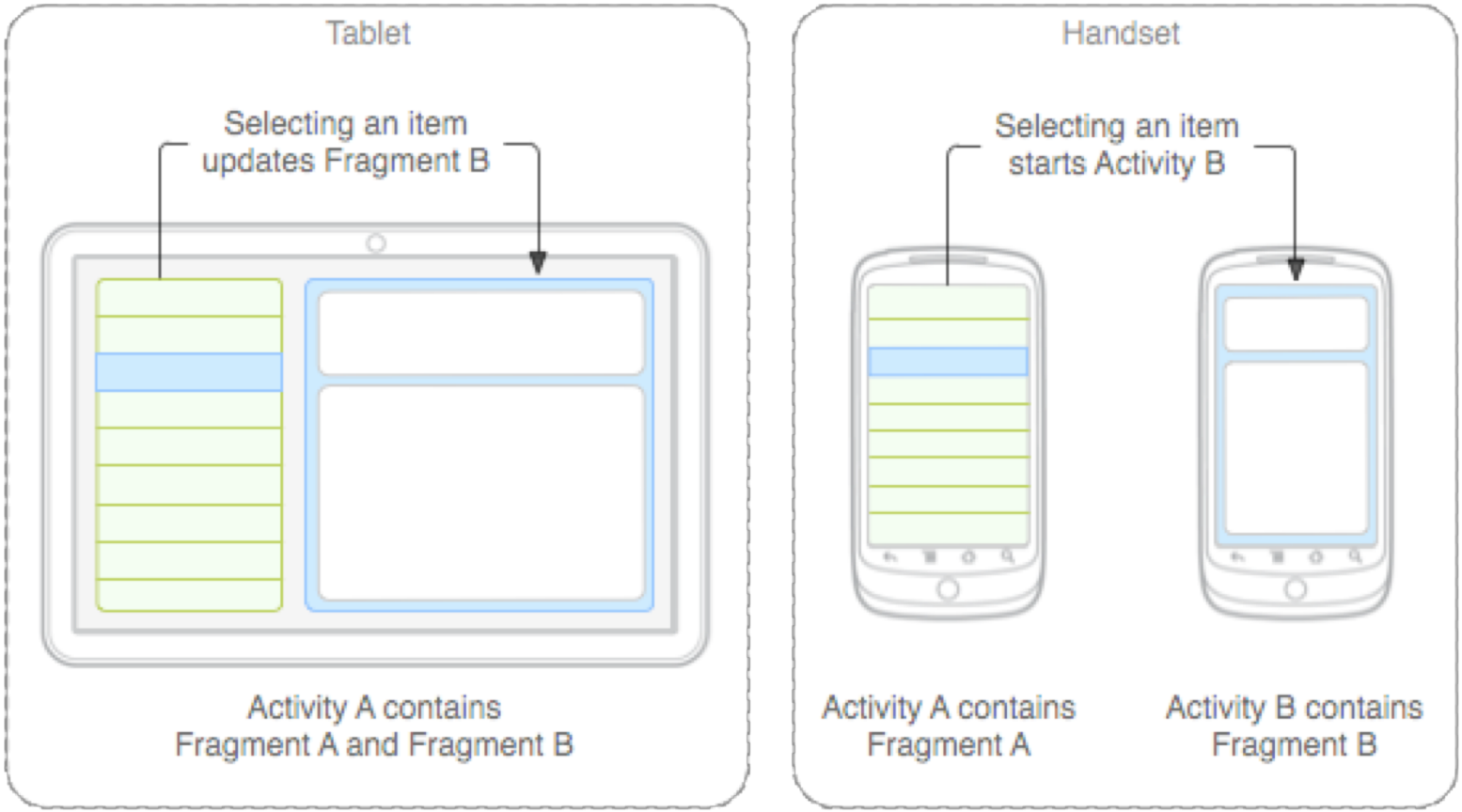
Example configuration

```
res/layout/my_layout.xml                 // layout for normal screen size ("default")
res/layout-large/my_layout.xml           // layout for large screen size
res/layout-xlarge/my_layout.xml          // layout for extra-large screen size
res/layout-xlarge-land/my_layout.xml     // layout for extra-large in landscape orientation

res/drawable-mdpi/graphic.png            // bitmap for medium-density
res/drawable-hdpi/graphic.png            // bitmap for high-density
res/drawable-xhdpi/graphic.png           // bitmap for extra-high-density
res/drawable-xxhdpi/graphic.png          // bitmap for extra-extra-high-density

res/mipmap-mdpi/my_icon.png            // launcher icon for medium-density
res/mipmap-hdpi/my_icon.png            // launcher icon for high-density
res/mipmap-xhdpi/my_icon.png           // launcher icon for extra-high-density
res/mipmap-xxhdpi/my_icon.png          // launcher icon for extra-extra-high-density
res/mipmap-xxxhdpi/my_icon.png         // launcher icon for extra-extra-extra-high-density
```

New problem


Tablet
Selecting an item
updates Fragment B
Activity A contains
Fragment A and Fragment B
Handset
Selecting an item
starts Activity B
Activity A contains
Fragment A
Activity B contains
Fragment B

Fragments

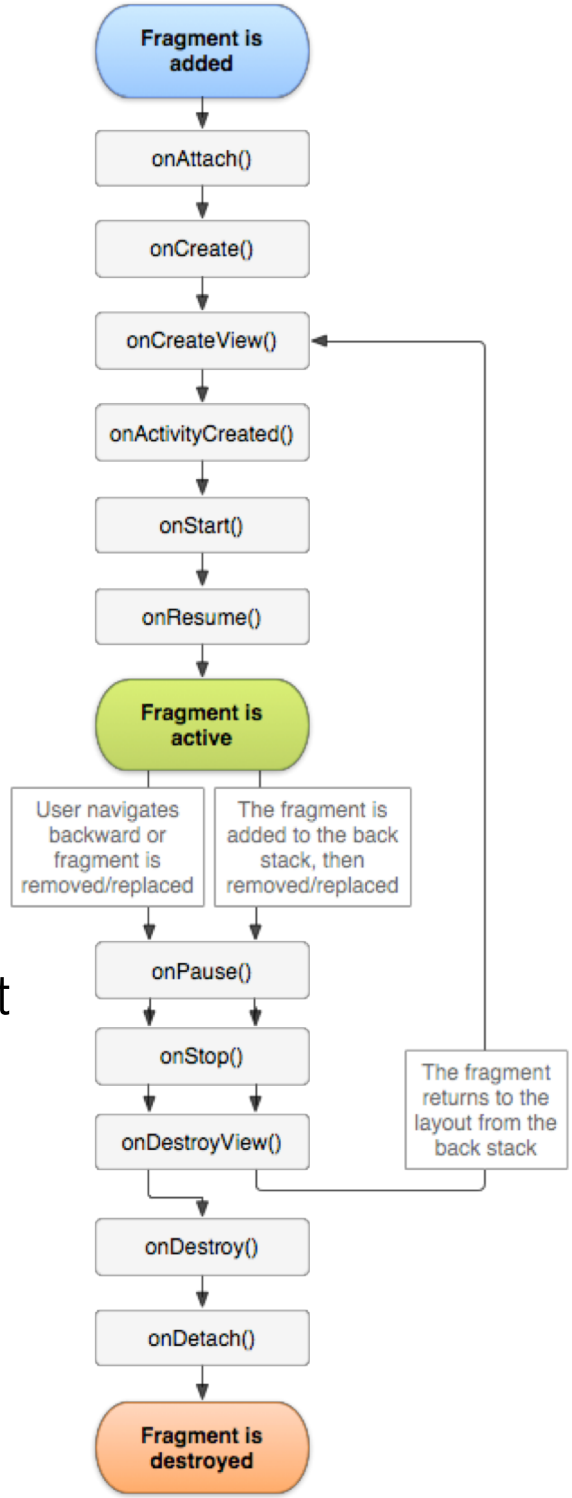
- Introduced in Android 3.0 (API level 11) to support more dynamic and flexible UI designs on large screens
- One can combine multiple fragments in a single activity to build a multi-pane UI and reuse the same fragments in multiple activities
- Represent behaviors or portions of user interface in Activities
  - Must always be embedded in an activity and their lifecycle is directly affected by the one of the host activity
  - Can be manipulated independently
- We may also use
  - A fragment without UI as invisible worker for the activity
  - Special-purpose fragments: DialogFragment, ListFragment, PreferenceFragment

Reuse

- You should design each fragment as a modular and reusable activity component
  - Each fragment defines its own layout and its own behavior
- You can include one fragment in multiple activities, so you should design for reuse
  - Avoid directly manipulating one fragment from another fragment
  - A modular fragment allows you to change your fragment combinations for different screen sizes

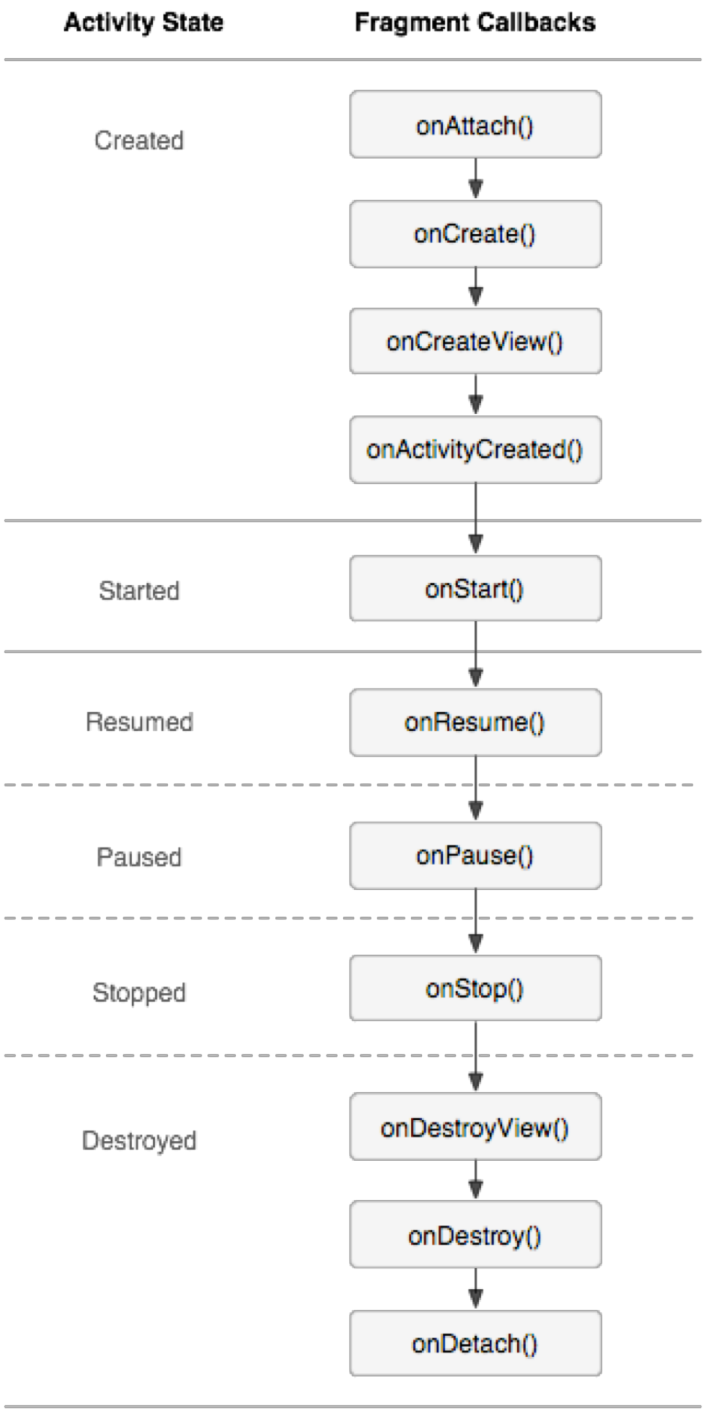
Fragment

- onCreate() initializes essential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed
- onCreateView() called when it's time for the fragment to draw its user interface for the first time
  - It returns a View that is the root of the fragment's layout
  - It can return null if the fragment does not provide a UI
- onPause() called when the user is leaving the fragment
  - This is usually where you should commit any changes that should be persisted
- A fragment can contribute menu items to the activity's Options Menu (and, consequently, the app bar)



Activities and fragments

- Lifecycle of the activity in which the fragment lives directly affects the lifecycle of fragment
- Some further callbacks
  - Each lifecycle callback for the activity results in a similar callback for each fragment
  - onAttach() called when fragment has been associated with activity
  - onActivityCreated() called when activity's onCreate() method has returned
  - onDestroyView() called when the view hierarchy associated with fragment is being removed
  - onDetach() is called when the fragment is being disassociated from activity



Example

```
public static class ExampleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater,
                             ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inf.inflate(R.layout.ex_frag, container, false);
    }
}
```

Adding fragments to activities

```
<LinearLayout xmlns:android=…
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:baselineAligned="false">
    <fragment android:name="it.polimi.first.ArticleListFragment"
            android:id="@+id/list"
            android:layout_weight="1"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
    <fragment android:name="it.polimi.first.ArticleReaderFragment"
            android:id="@+id/viewer"
            android:layout_weight="2"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
</LinearLayout>
```

FragmentManager

- Needed to Manage fragments in an activity
- getFragmentManager() provides a manager that can then be used to
  - Add, remove, replace fragments (through transactions)
  - Get fragments that exist in the activity
  - Pop fragments off the back stack
  - Register a listener for changes to the back stack

Adding a fragment

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

ExampleFragment fragment = new ExampleFragment();
fragmentTransaction.add(R.id.fragment_container, fragment);
fragmentTransaction.commit();
```

- To add a fragment without a UI, add the fragment using add(Fragment, String)
  - Supply a unique string "tag", instead of a view ID
  - onCreateView() is not called

addToBackStack()

```
// Create new fragment and transaction

Fragment newFragment = new ExampleFragment();
FragmentTransaction transaction =
                    getFragmentManager().beginTransaction();

// Replace whatever is in the fragment_container view with
// this fragment, and add the transaction to the back stack

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

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

Fragments and Activities

- A given instance of a fragment is directly tied to the activity that contains it
  - The fragment can then access the activity instance with getActivity() and easily perform tasks such as find a view in the activity
- Likewise the activity can call methods in the fragment by acquiring a reference to the fragment from the FragmentManager

Menus

Three options (I)

- Android apps should provide an app bar to present common user actions

- **Options menu and app bar**
  - Is the primary collection of menu items (actions that have a global impact) for an activity
  - On Android 3.0 and higher, these items are presented by the app bar as a combination of on-screen action items and overflow options

Three options (II)

- **Context menu and contextual action mode**
  - It is a floating menu that appears when the user performs a long-click on an element and provides actions that affect the selected content or context frame
  - On Android 3.0 and higher, you should instead use the contextual action mode to display action items in a bar at the top of the screen and allow the user to select multiple items
- **Popup menu**
  - It displays a list of items in a vertical list that is anchored to the view that invoked the menu
    - Good for providing an overflow of actions that relate to specific content

Menus

- Must be properly designed
  - If they become too big, they do not fit
  - Should guide the user properly
  - Should exploit the current state/context
- Two ways
  - XML: file in res/menu and inflate it (reuse)
  - Java: directly part of the activity's code

Usual XML definition

- Each menu entry is specified by an ID, a label, and an icon (optional)
  - Method onCreateOptionsMenu() inflates the menu
    - Specific actions must then be implemented
  - One can add a submenu to an item in any menu (except a submenu)
  - A menu group is a collection of menu items that share properties

```
<menu xmlns:android= …>
    <item android:id="@+id/newgame"
          android:icon="@drawable/ic_launcher"
          android:title="New Game"
          android:showAsAction="ifRoom"/>
    <item android:id="@+id/help"
          android:icon="@drawable/ic_launcher"
          android:title="Help" />
</menu>
```

Within the activity

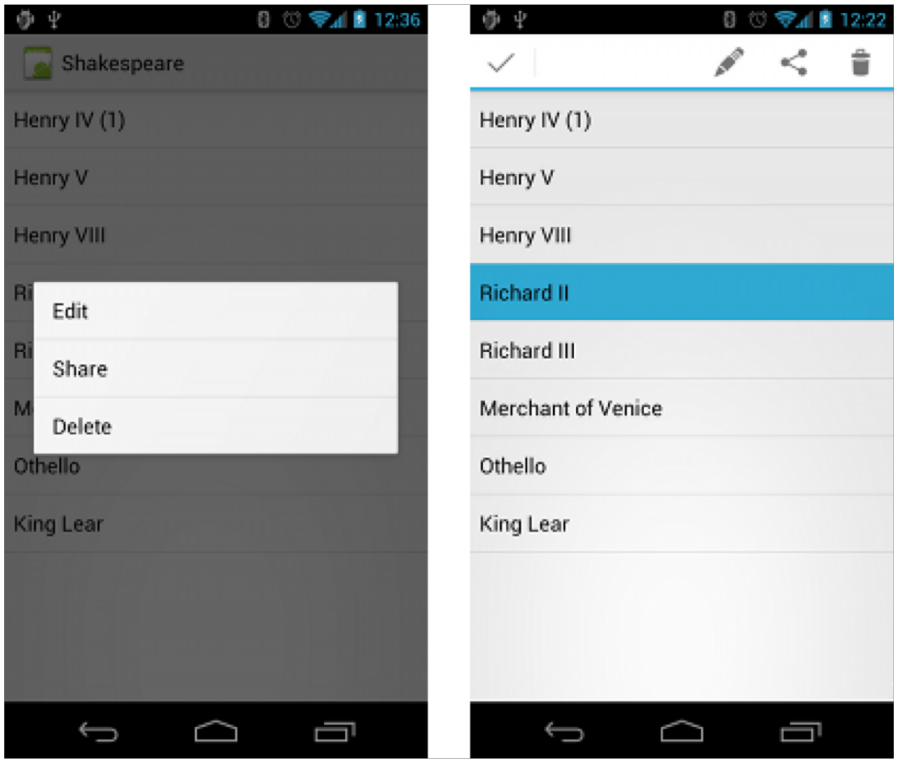
```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.main, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
        case R.id.newgame:
            newGame();
            return true;
        case R.id.help:
            showHelp();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Contextual menus

- Most often used for items in a ListView, GridView, or other view collections in which the user can perform direct actions on each item

floating context menu



contextual action bar

Popup menus

- Instantiate a PopupMenu with its constructor
  - Current application Context and the View to which the menu should be anchored as parameters
- Use MenuInflater to inflate your menu resource into the Menu object returned by PopupMenu.getMenu()
- Call PopupMenu.show()

```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_overflow_
    android:contentDescription="@string/
    android:onClick="showPopup" />
```

```
public void showPopup(View v) {
    PopupMenu popup = new PopupMenu(this, v);
    MenuInflater inflater = popup.getMenuInflater();
    inflater.inflate(R.menu.actions, popup.getMenu());
    popup.show();
}
```

Another example

```
public void showMenu(View v) {
    PopupMenu popup = new PopupMenu(this, v);

    // This activity implements OnMenuItemClickListener
    popup.setOnMenuItemClickListener(this);
    popup.inflate(R.menu.actions);
    popup.show();
}

@Override
public boolean onMenuItemClick(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.archive:
            archive(item);
            return true;
        case R.id.delete:
            delete(item);
            return true;
        default:
            return false;
    }
}
```