Android Developer Fundamentals

User Interaction and Navigation

Lesson 4

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

4.1 User Input Controls

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Contents

- User Interaction
- Focus
- Text input and keyboards
- Radio Buttons and Checkboxes
- Making Choices
  - dialogs, spinners and pickers
- Recognizing gestures

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

User Interaction

Users expect to interact with apps

- Clicking, pressing, talking, typing, and listening
- Using user input controls such buttons, menus, keyboards, text boxes, and a microphone
- Navigating between activities

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

User interaction design

Important to be obvious, easy, and consistent:

- Think about how users will use your app
- Minimize steps
- Use UI elements that are easy to access, understand, use
- Follow Android best practices
- Meet user's expectations

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Input Controls

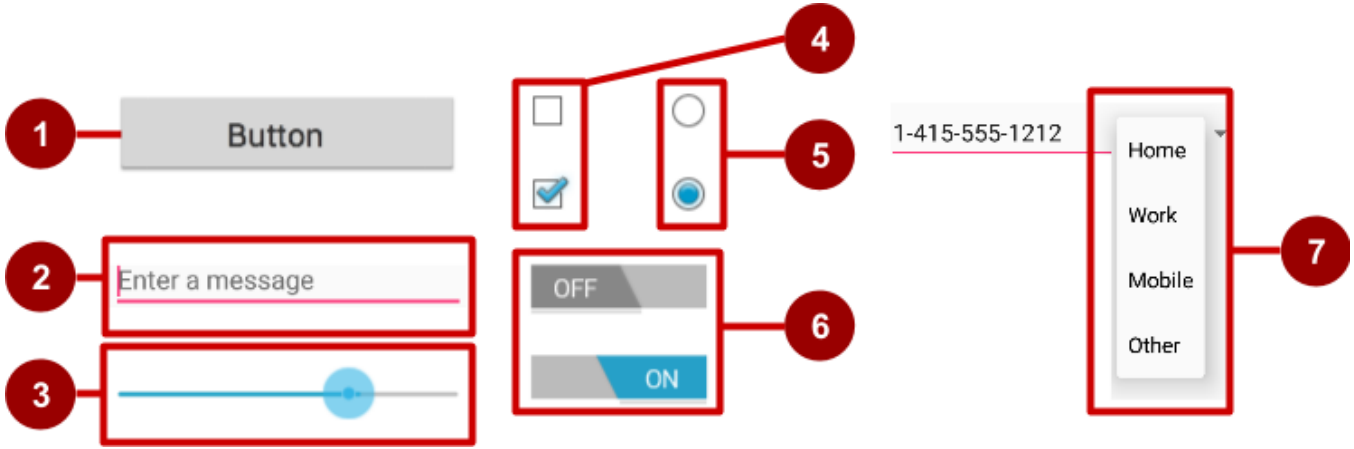
Ways to get input from the user

- Free form
  - Text and voice input
- Actions
  - Buttons
  - Contextual menus
  - Gestures
  - Dialogs
- Constrained choices
  - Pickers
  - Checkboxes
  - Radio buttons
  - Toggle buttons
  - Spinners

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

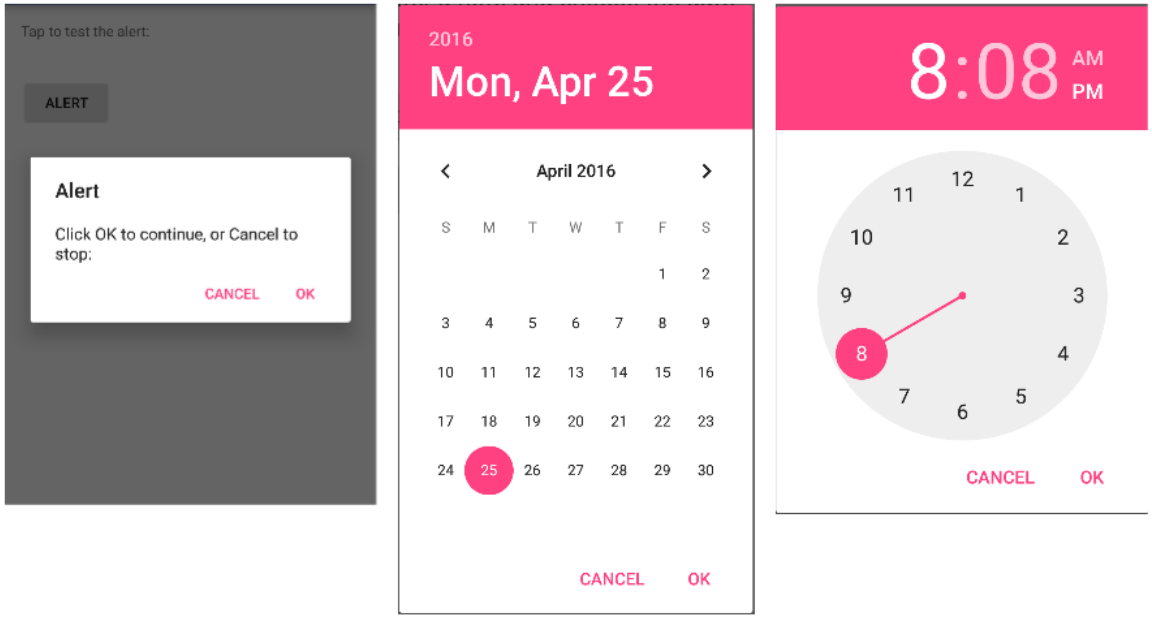
Examples of user input controls

1. Button
2. Text field
3. Seek bar
4. Checkboxes
5. Radio buttons
6. Toggle
7. Spinner



This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Alert dialog, date picker, time picker



This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

View is base class for input controls

- The View class is the basic building block for all UI components, including input controls
- View is the base class for classes that provide interactive UI components
- View provides basic interaction through `android:onClick`

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Focus

Focus

- The view that receives user input has "Focus"
- Only one view can have focus
- Focus makes it unambiguous which view gets the input
- Focus is assigned by
  - User tapping a view
  - App guiding the user from one text input control to the next using the Return, Tab, or arrow keys
  - Calling `requestFocus()` on any view that is focusable

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Clickable versus focusable

Clickable—View can respond to being clicked or tapped

Focusable—View can gain focus to accept input

Input controls such as keyboards send input to the view that has focus

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Which View gets focus next?

- Topmost view under the touch
- After user submits input, focus moves to nearest neighbor—priority is left to right, top to bottom
- Focus can change when user interacts with a directional control

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Guiding users

- Visually indicate which view has focus so users knows where their input goes
- Visually indicate which views can have focus helps users navigate through flow
- Predictable and logical—no surprises!

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Guiding focus

- Arrange input controls in a layout from left to right and top to bottom in the order you want focus assigned
- Place input controls inside a view group in your layout
- Specify ordering in XML

```
android:id="@+id/top"
android:focusable="true"
android:nextFocusDown="@+id/bottom"
```


This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Set focus explicitly

Use methods of the View class to set focus

- setFocusable() sets whether a view can have focus
- requestFocus() gives focus to a specific view
- setOnFocusChangeListener() sets listener for when view gains or loses focus
- onFocusChanged() called when focus on a view changes

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Find the view with focus

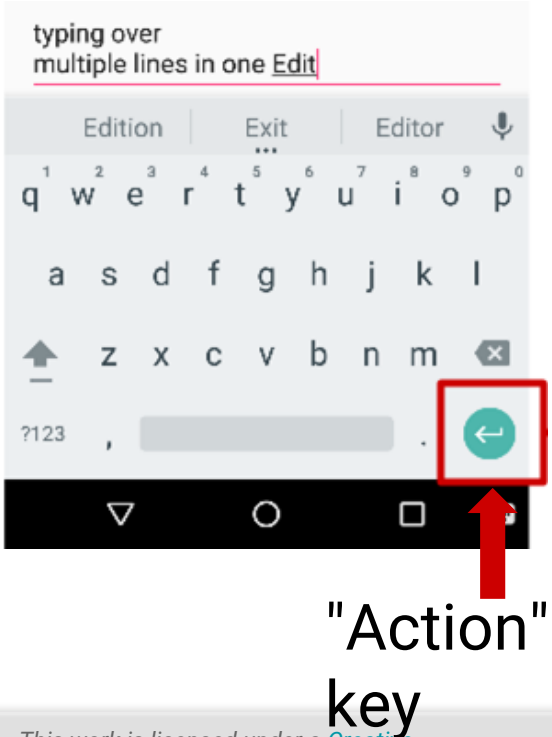
- Activity.getCurrentFocus()
- ViewGroup.getFocusedChild()

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Text Input

EditText

- EditText class
- Multiple lines of input
- Characters, numbers, and symbols
- Spelling correction
- Tapping the Return (Enter) key starts a new line
- Customizable



This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Getting text

- Get the EditText object for the EditText view

```
EditText simpleEditText =
    (EditText) findViewById(R.id.edit_simple);
```

- Retrieve the CharSequence and convert it to a string

```
String strValue =
    simpleEditText.getText().toString();
```


This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Common input types

- textShortMessage—Limit input to 1 line
- textCapSentences—Set keyboard to caps at beginning of sentences
- textAutoCorrect—Enable autocorrecting
- textPassword—Conceal typed characters
- textEmailAddress—Show an @ sign on the keyboard
- phone—numeric keyboard for phone numbers

```
android:inputType="phone"

android:inputType="textAutoCorrect|textCapSentences"
```


This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Buttons

Button

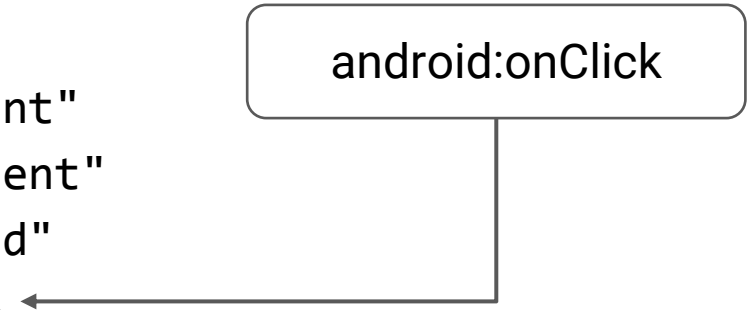
- View that responds to clicking or pressing
- Usually text or visuals indicate what will happen when it is pressed
- Views: Button > ToggleButton, ImageView > FloatingActionButton (FAB)
- State: normal, focused, disabled, pressed, on/off
- Visuals: raised, flat, clipart, images, text

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Responding to button taps

- *In your code*: Use `OnClickListener` event listener.
- *In XML*: use `android:onClick` attribute in the XML layout:

```
<Button
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```



This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Setting listener with onClick callback

```
Button button = (Button) findViewById(R.id.button);

button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Do something in response to button click
    }
});
```

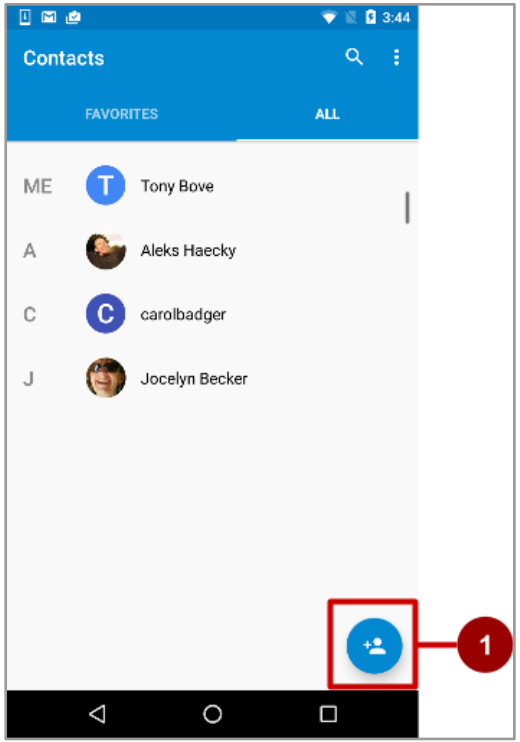

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Floating Action Buttons (FAB)

- Raised, circular, floats above layout
- Primary or "promoted" action for a screen
- One per screen

For example:

Add Contact button in Contacts app



This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Using FABs

- Add design support library to build.gradle

```
compile 'com.android.support:design:a.b.c'
```

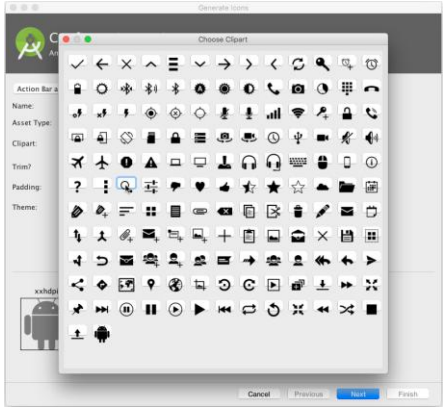
- Layout

```
<android.support.design.widget.FloatingActionButton
        android:id="@+id/fab"
        android:layout_gravity="bottom|end"
        android:layout_margin="@dimen/fab_margin"
        android:src="@drawable/ic_fab_chat_button_white"
        .../>
```


This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Button image assets

1. Right-click app/res/drawable
2. Choose **New > Image Asset**
3. Choose **Action Bar and Tab Items** from drop down menu
4. Click the **Clipart:** image (the Android logo)



Experiment:

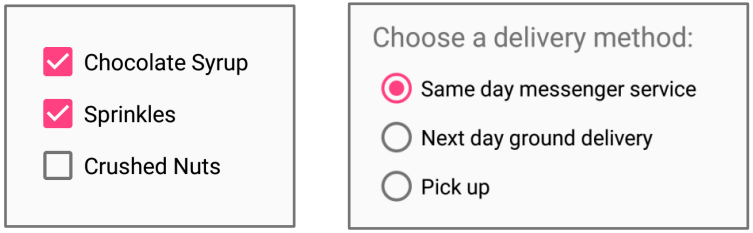
2. Choose **New > Vector Asset**

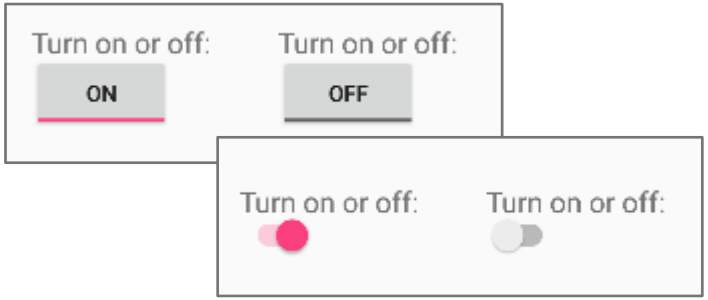
This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

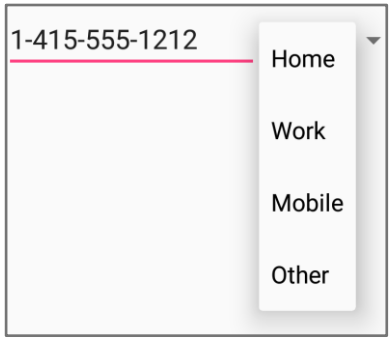
Making Choices

So many choices!

- Checkboxes
- Radio buttons
- Toggles
- Spinner





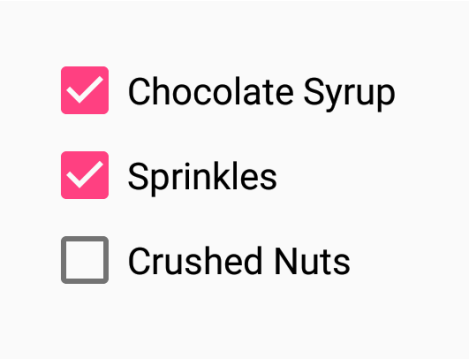


This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Checkboxes, radio buttons and toggles

Checkboxes

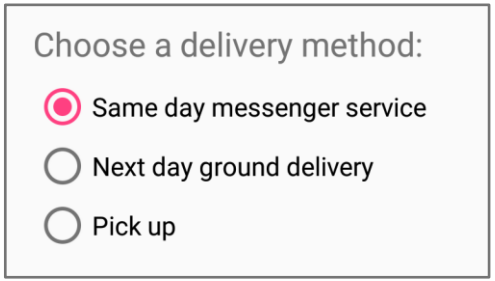
- User can select any number of choices
- Checking one box does not uncheck another
- Users expect checkboxes in a vertical list
- Commonly used with a submit button
- Every checkbox is a view and can have an onClick handler



This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Radio buttons

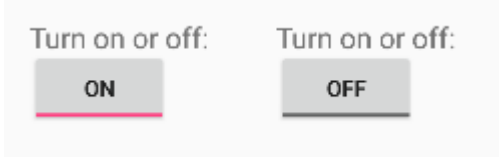
- User can select one of a number of choices
- Put radio buttons in a RadioGroup
- Checking one unchecks another
- Put radio buttons in a vertical list
  or horizontally if labels are short
- Every radio button can have an onClick handler
- Commonly used with a submit button
  for the RadioGroup



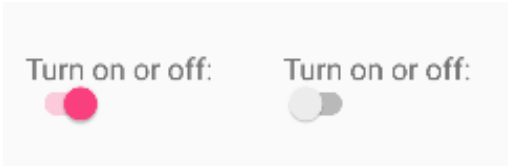
This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Toggle buttons and switches

- User can switch between 2 exclusive states (on/off)
- Use `android:onClick`+callback—or handle clicks in code



Toggle buttons



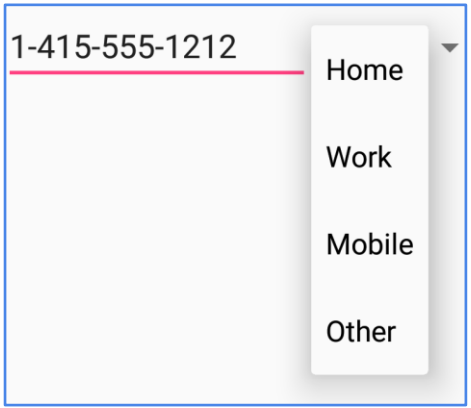
Switches

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Spinners

Spinners

- Quick way to select value from a set
- Drop-down list shows all values, user can select only one
- Spinners scroll automatically if necessary
- Use the `Spinner` class.



This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Implementing a spinner

1. Create Spinner UI element in the XML layout
2. Define spinner choices in an array
3. Create Spinner and set onItemSelectedListener
4. Create an adapter with default spinner layouts
5. Attach the adapter to the spinner
6. Implement `onItemSelectedListener` method

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Create spinner XML

In layout XML file

```
<Spinner
    android:id="@+id/label_spinner"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    android:entries="@array/array_name"
</Spinner>
```


This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Define array of spinner choices

In arrays.xml resource file

```
<string-array name="labels_array">
    <item>Home</item>
    <item>Work</item>
    <item>Mobile</item>
    <item>Other</item>
</string-array>
```


This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Create spinner and attach listener

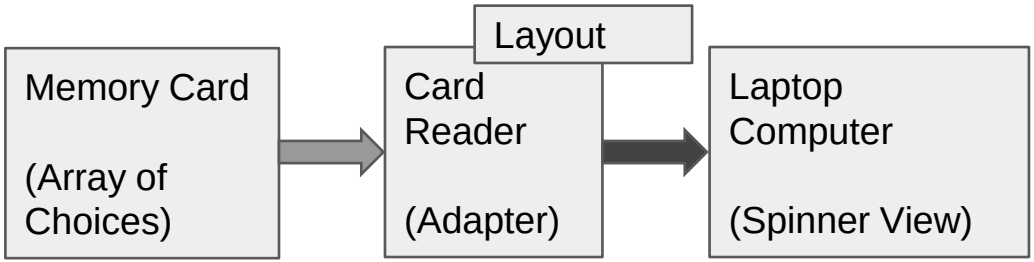
```
public class MainActivity extends AppCompatActivity implements
AdapterView.OnItemSelectedListener

// In onCreate()

Spinner spinner = (Spinner) findViewById(R.id.label_spinner);
if (spinner != null) {
      spinner.setOnItemSelectedListener(this);
}
```


This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

What is an adapter?

An adapter is like a bridge, or intermediary, between two incompatible interfaces

For example, a memory card reader acts as an adapter between the memory card and a laptop



This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Create adapter

Create ArrayAdapter using string array and default spinner layout

```
ArrayAdapter<CharSequence> adapter =
  ArrayAdapter.createFromResource(
        this, R.array.labels_array,
        // Layout for each item
        android.R.layout.simple_spinner_item);
```


This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Attach the adapter to the spinner

- Specify the layout for the drop down menu

```
adapter.setDropDownViewResource(
    android.R.layout.simple_spinner_dropdown_item);
```

- Attach the adapter to the spinner

```
spinner.setAdapter(adapter);
```


This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Implement onItemSelectedListener

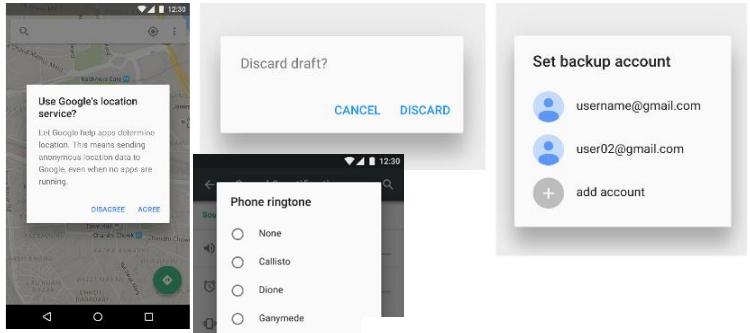
```
public class MainActivity extends AppCompatActivity implements
AdapterView.OnItemSelectedListener

    public void onItemSelected(AdapterView<?> adapterView,
        View view, int pos, long id) {
           String spinner_item =
             adapterView.getItemAtPosition(pos).toString();
           // Do something here with the item
    }

    public void onNothingSelected(AdapterView<?> adapterView) {
       // Do something
    }
```


This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

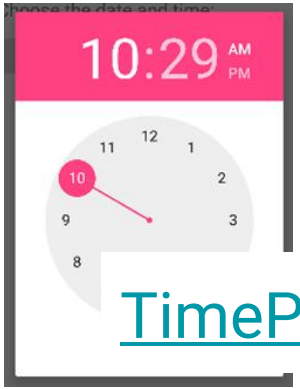
Dialogs

Dialogs

- Dialog appears on top, interrupting the flow of activity
- Require user action to dismiss



AlertDialog



TimePickerDialog

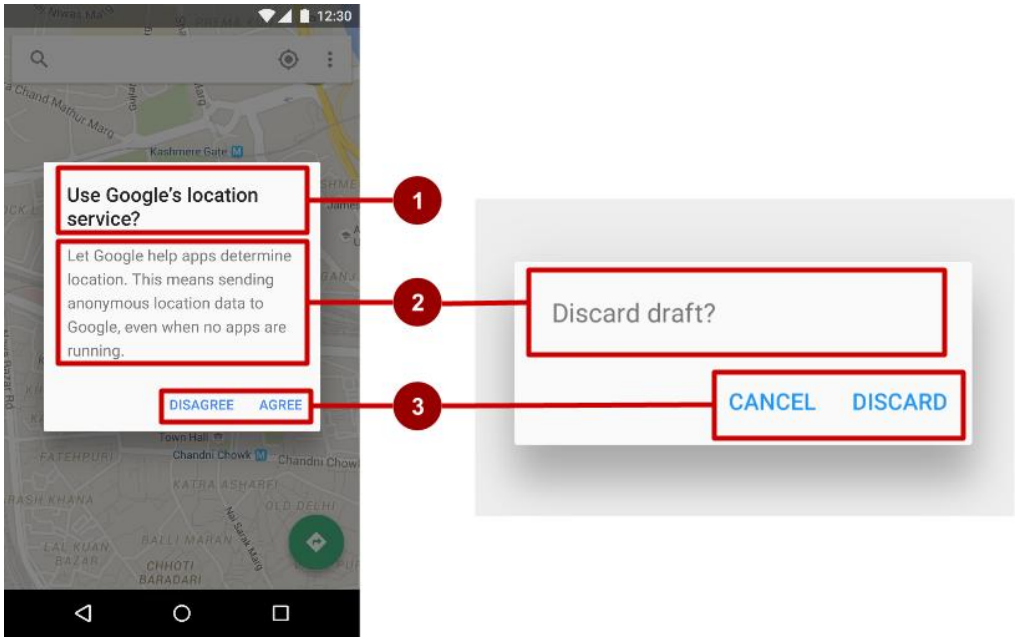
DatePickerDialog

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

AlertDialog

AlertDialog can show:

1. Title (optional)
2. Content area
3. Action buttons



This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Build the AlertDialog

Use `AlertDialog.Builder` to build a standard alert dialog and set attributes:

```
public void onClickShowAlert(View view) {
    AlertDialog.Builder alertDialog = new
                        AlertDialog.Builder(MainActivity.this);
    alertDialog.setTitle("Connect to Provider");
    alertDialog.setMessage(R.string.alert_message);
    ...
```


This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Add the button actions

- `alertDialog.setPositiveButton()`
- `alertDialog.setNeutralButton()`
- `alertDialog.setNegativeButton()`

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

alertDialog code example

```
alertDialog.setPositiveButton(
    "OK", newDialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
        // User clicked OK button.
    }
});
```

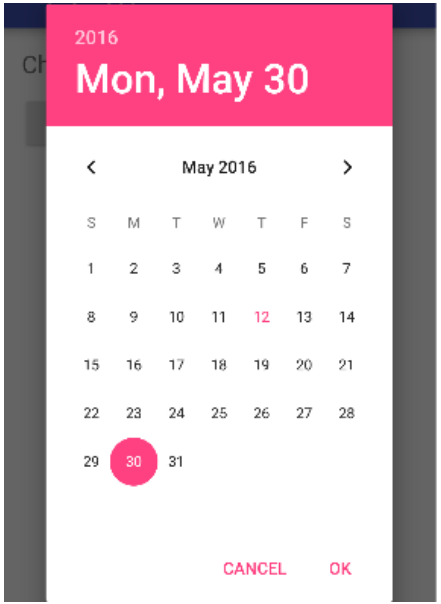
Same pattern for setNegativeButton() and setNeutralButton()

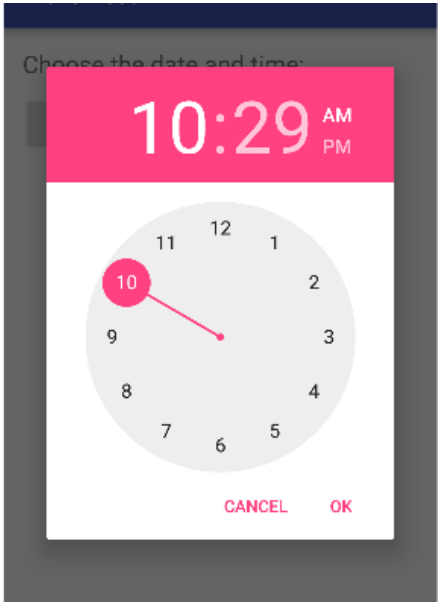
This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Pickers

Pickers

- DatePickerDialog
- TimePickerDialog

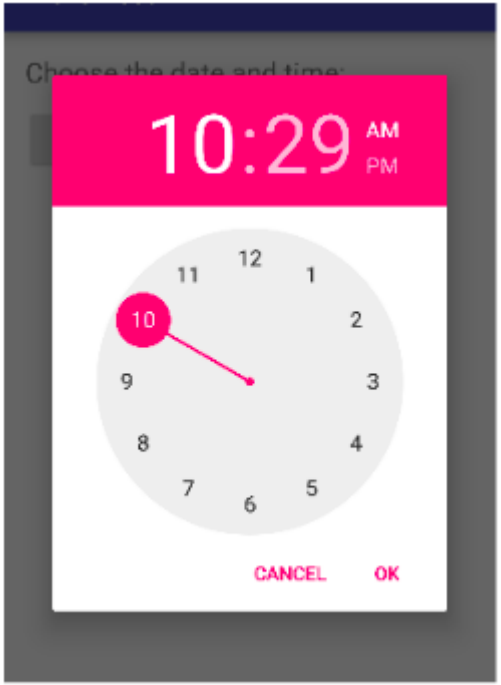




This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Pickers use fragments

- Use DialogFragment to show a picker
- DialogFragment is a window that floats on top of activity's window



This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Introduction to fragments

- A fragment is like a mini-activity within an activity
  - Manages its own own lifecycle.
  - Receives its own input events.
- Can be added or removed while parent activity is running
- Multiple fragments can be combined in a single activity
- Can be reused in multiple activities

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Creating a date picker dialog

1. Add a blank fragment that extends `DialogFragment` and implements `DatePickerDialog.OnDateSetListener`
2. In `onCreateDialog()` initialize the date and return the dialog
3. In `onDateSet()` handle the date
4. In Activity show the picker and add a method to use the date

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Creating a time picker dialog

1. Add a blank fragment that extends `DialogFragment` and implements `TimePickerDialog.OnTimeSetListener`
2. In `onCreateDialog()` initialize the time and return the dialog
3. In `onTimeSet()` handle the time
4. In Activity, show the picker and add a method to use the time

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Common Gestures

Touch Gestures

Touch gestures include:

- long touch
- double-tap
- fling
- drag
- scroll
- pinch

Don’t depend on touch gestures for app's basic behavior!

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Detect gestures

Classes and methods are available to help you handle gestures.

- GestureDetectorCompat class for common gestures
- MotionEvent class for motion events

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Detecting all types of gestures

1. Gather data about touch events.
2. Interpret the data to see if it meets the criteria for any of the gestures your app supports.

Read more about how to handle gestures in the Android developer documentation

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Learn more

- Input Controls
- Drawable Resources
- Floating Action Button
- Radio Buttons
- Specifying the Input Method Type
- Handling Keyboard Input
- Text Fields
- Buttons
- Spinners
- Dialogs
- Fragments
- Input Events
- Pickers
- Using Touch Gestures
- Gestures design guide

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

What's Next?

- Concept Chapter: 4.1 C User Input Controls
- Practical:
  4. P Using Keyboards, Input Controls, Alerts, and Pickers

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

END

Android Developer Fundamentals

User Interaction and Intuitive Navigation

Lesson 4

4.2 Menus

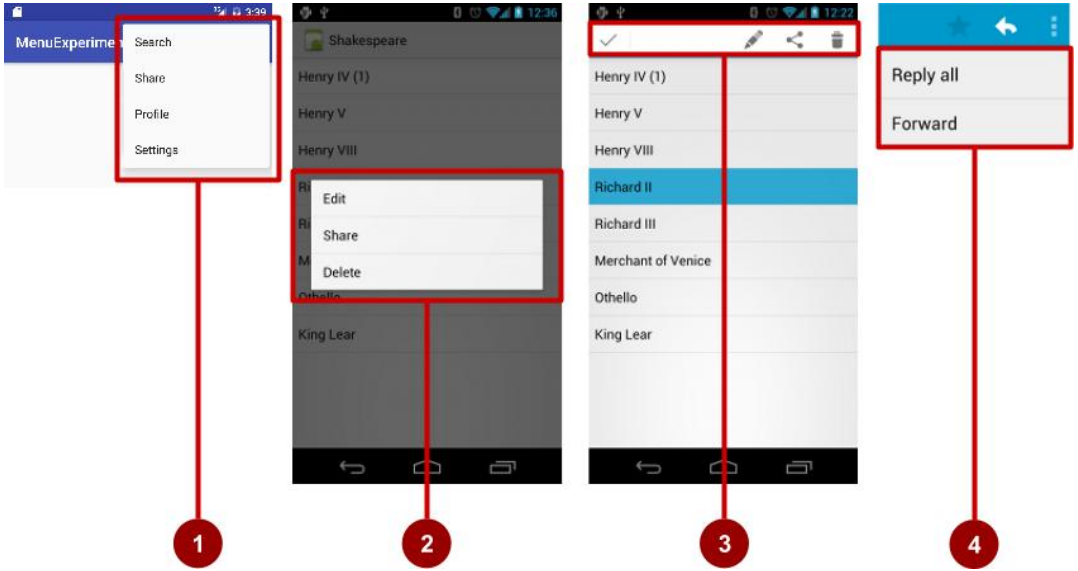
This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Contents

- App Bar with Options Menu
- Contextual menus
- Popup menus

Types of Menus

1. App bar with options menu
2. Contextual menu
3. Contextual action bar
4. Popup menu

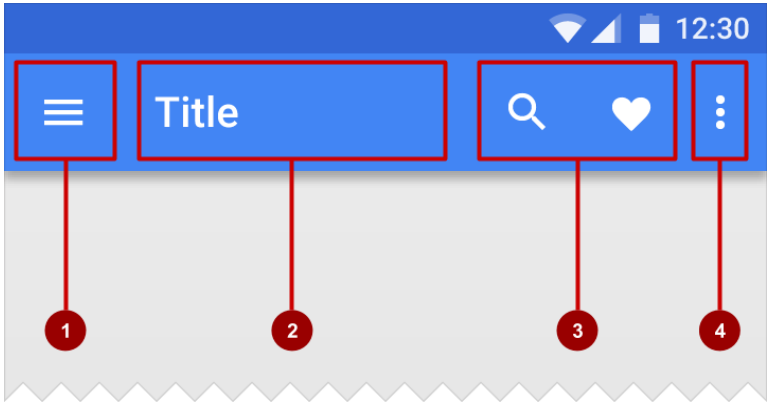


App Bar with Options Menu

What is the App Bar?

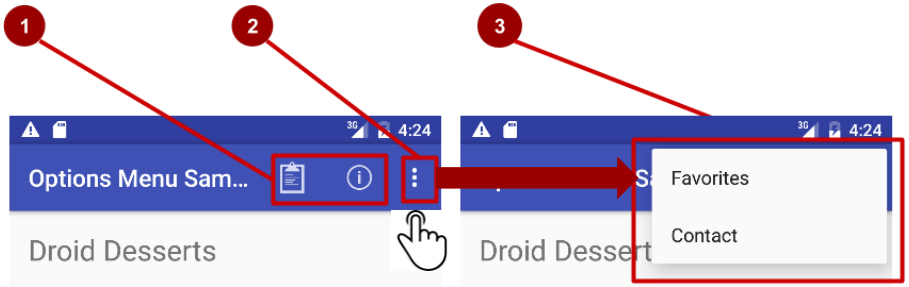
Bar at top of each screen—(usually) the same for all screens

1. Nav icon to open navigation drawer
2. Title of current activity
3. Icons for options menu items
4. Action overflow button for the rest of the options menu



What is the options menu?

- Action icons in the app bar for important items (1)
- Tap the three dots, the "action overflow button" to see the options menu (2)

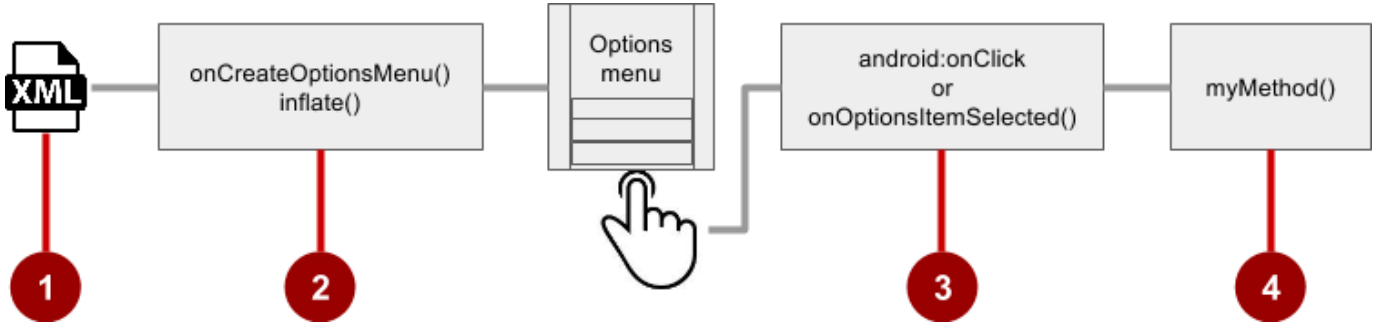


- Appears in the right corner of the app bar (3)
- For navigating to other activities and editing app settings

Adding Options Menu

Steps to implement options menu

1. XML menu resource (`menu_main.xml`)
2. `onCreateOptionsMenu()` to inflate the menu
3. `onClick` attribute or `onOptionsItemSelected()`
4. Method to handle item click



Create menu resource

1. Create menu resource directory
2. Create XML menu resource (`menu_main.xml`)
3. Add an entry for each menu item

```
<item  android:id="@+id/option_settings"
       android:title="@string/settings" />

<item  android:id="@+id/option_toast"
       android:title="@string/toast" />
```

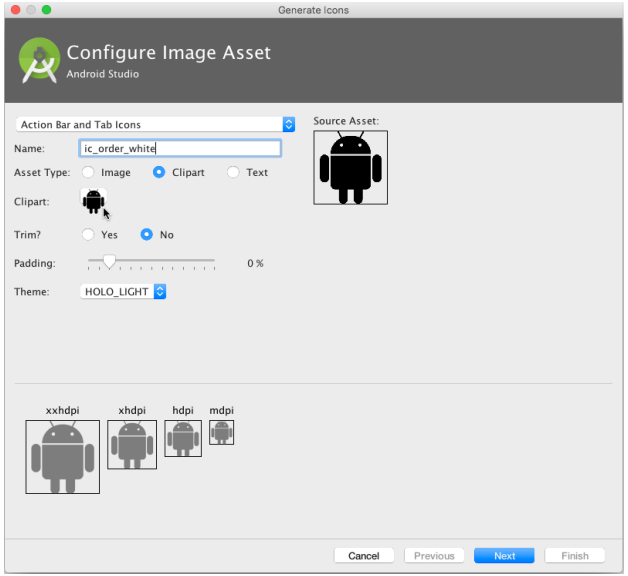
Inflate options menu

- Override `onCreateOptionsMenu()` in main activity

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

Add icons for menu items

1. Right-click **drawable**
2. Choose **New > Image Asset**
3. Choose **Action Bar and Tab Items**
4. Edit the icon name
5. Click clipart image, and click icon
6. Click **Next**, then **Finish**



Add menu item attributes

```
<item android:id="@+id/action_order"
      android:icon="@drawable/ic_toast_dark"
      android:title="@string/toast"
      android:titleCondensed="@string/toast_condensed"
      android:orderInCategory="1"
      app:showAsAction="ifRoom" />
```

Override onOptionsItemSelected()

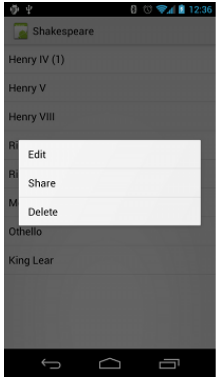
```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_order:
            showOrder();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

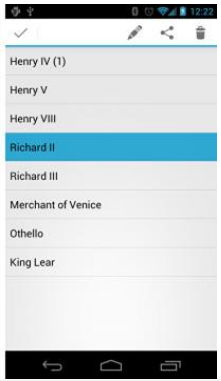
Contextual Menus

What are contextual menus?

- Allow users to perform an action on a selected view or content
- Can be deployed on any View object, but most often used for items in a RecyclerView, GridView, or other view collection

Types of contextual menus

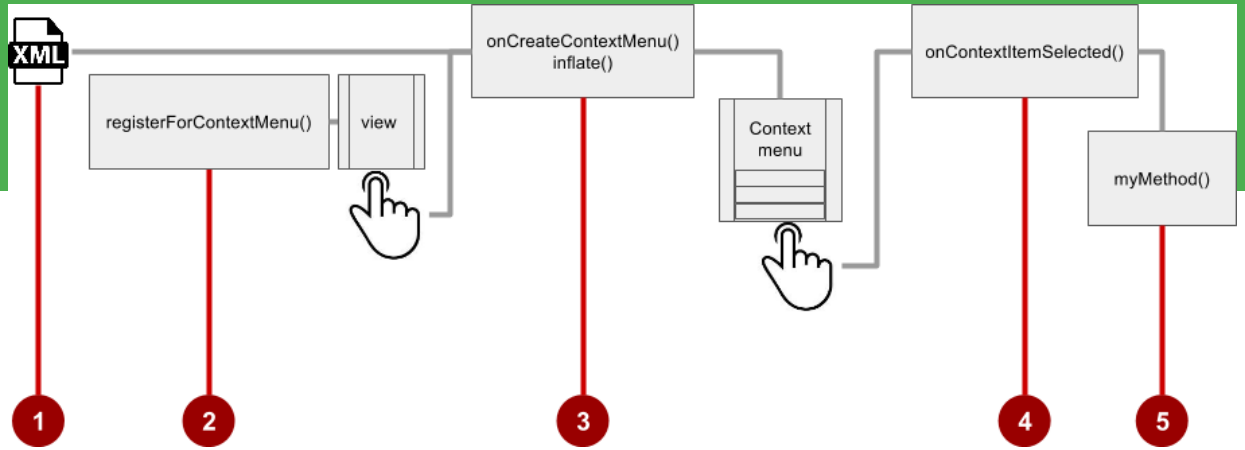




- Floating context menu—floating list of menu items when long-presses on a view element
  - User can modify the view element or use it in some fashion
  - Users perform a contextual action on one view element at a time
- Contextual action mode—temporary action bar in place of or underneath the app bar
  - Action items affect the selected view element(s)
  - Users can perform action on multiple view elements at once

Floating Context Menu

Steps



1. Create XML menu resource file and assign appearance and position attributes
2. Register view to use a context menu using `registerForContextMenu()`
3. Implement `onCreateContextMenu()` in the activity or fragment to inflate the menu
4. Implement `onContextItemSelected()` to handle menu item clicks
5. Create a method to perform an action for each context menu item

Create menu resource

- Create XML menu resource (`menu_context.xml`)

```
<item
    android:id="@+id/context_edit"
    android:title="@string/edit"
    android:orderInCategory="10"/>

<item
    android:id="@+id/context_share"
    android:title="@string/share"
    android:orderInCategory="20"/>
```

Register a view to a context menu

- in onCreate() of the activity
- registers View.OnCreateContextMenuListener
- Does not specify which context menu!

```
TextView article_text = (TextView) findViewById(R.id.article);
registerForContextMenu(article_text);
```

Implement onCreateContextMenu()

Specifies which context menu

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenu.ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu_context, menu);
}
```

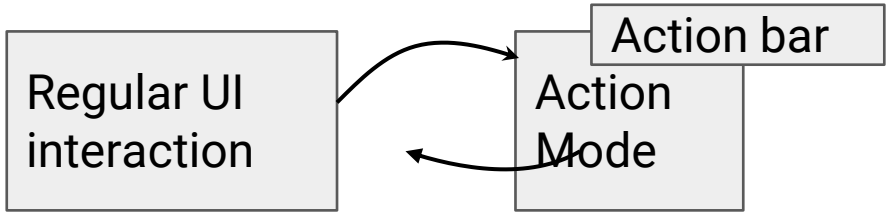
Implement onContextItemSelected()

```
@Override
public boolean onContextItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.context_edit:
            editNote();
            return true;
        default:
            return super.onContextItemSelected(item);
    }
}
```

Contextual Action Bar

What is Action Mode?

- ActionMode is a UI mode that lets you replace parts of the normal UI interactions temporarily
- For example, selecting a section of text or long-pressing an item could trigger action mode



Action mode has a lifecycle

- Start it with startActionMode(), for example, in the listener
- ActionMode.Callback interface provides the lifecycle methods that you can override
  - `onCreateActionMode(ActionMode, Menu)` once on initial creation
  - `onPrepareActionMode(ActionMode, Menu)` after creation and any time ActionMode is invalidated
  - `onActionItemClicked(ActionMode, MenuItem)` any time a contextual action button is clicked
  - `onDestroyActionMode(ActionMode)` when the action mode is closed

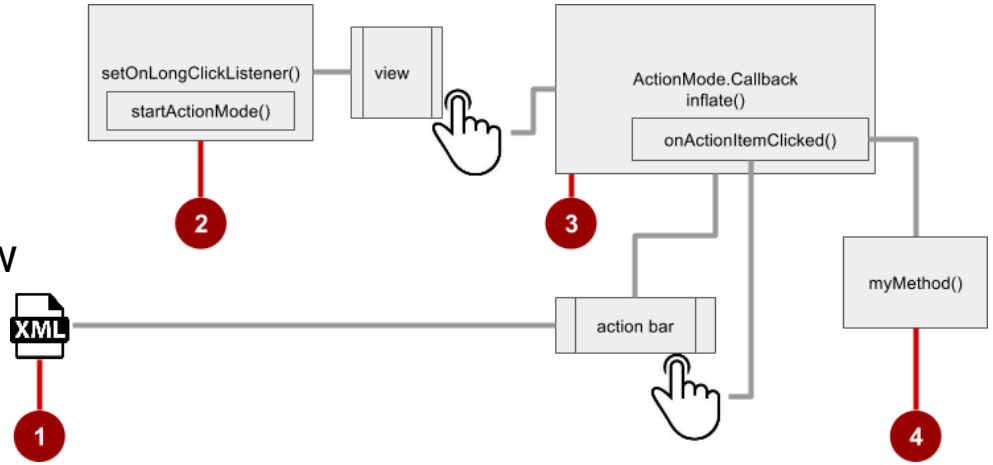
What is a contextual action bar?

Long-tap on the view shows contextual action bar

1. Contextual action bar with actions
   - Edit, Share, and Delete
   - Done (left arrow icon) on the left side
2. View on which long press triggers the contextual action bar
   - Action bar is available until user taps Done

Steps for contextual action bar

1. Create XML menu resource file and assign icons for items
2. `setOnLongClickListener()` on view that triggers the contextual action bar and call `startActionMode()` to handle click
3. Implement `ActionMode.Callback` interface to handle ActionMode lifecycle; include action for a menu item click in `onActionItemClicked()` callback
4. Create a method to perform an action for each context menu item



Use setOnLongClickListener

```
private ActionMode mActionMode;
```

In onCreate

```
View view = findViewById(article);
view.setOnLongClickListener(new View.OnLongClickListener() {
   public boolean onLongClick(View view) {
      if (mActionMode != null) return false;
      mActionMode =
            MainActivity.this.startActionMode(mActionModeCallback);
      view.setSelected(true);
      return true;
   }
});
```

Implement mActionModeCallback

```
public ActionMode.Callback mActionModeCallback =
    new ActionMode.Callback() {
     // Implement action mode callbacks here
};
```

Implement onCreateActionMode

```
@Override
public boolean onCreateActionMode(ActionMode mode, Menu menu) {
    MenuInflater inflater = mode.getMenuInflater();
    inflater.inflate(R.menu.menu_context, menu);
    return true;
}
```

Implement onPrepareActionMode

- Called each time the action mode is shown
- Always called after `onCreateActionMode`, but may be called multiple times if the mode is invalidated

```
@Override
public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
    return false; // Return false if nothing is done.
}
```

Implement onActionItemClicked

- Called when users selects an action
- Handle clicks in this method

```
@Override
public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
   switch (item.getItemId()) {
       case R.id.goodbyetextview:
         Toast.makeText(getApplicationContext(), "Menu Toast",
Toast.LENGTH_SHORT).show();
         mode.finish(); // Action picked, so close the action bar
         return true;
       default:
         return false;
   }
}
```

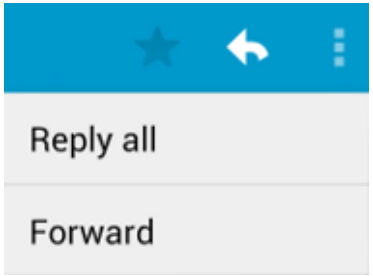
Implement onDestroyActionMode

- Called when user exits the action mode

```
@Override
public void onDestroyActionMode(ActionMode mode) {
    mActionMode = null;
}
```

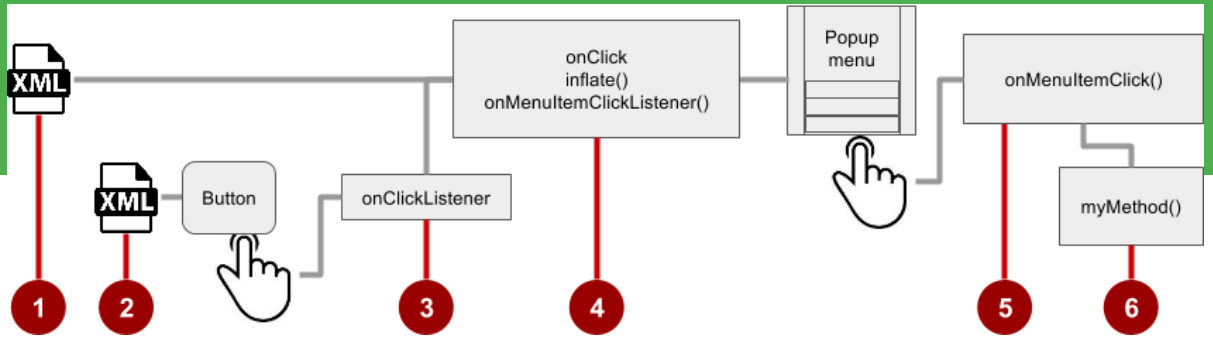
Popup Menu

What is a popup menu?



- Vertical list of items anchored to a view
- Typically anchored to a visible icon
- Actions should not directly affect view content
  - The options menu overflow that opens Settings
  - For example, in an email app, Reply All and Forward are related to the email message, but don't affect or act on the message

Steps



1. Create XML menu resource file and assign appearance and position attributes
2. Add an `ImageButton` for the popup menu icon in the XML activity layout file
3. Assign `onClickListener` to the button
4. Override `onClick()` to inflate the popup and register it with `onMenuItemClickListener()`
5. Implement `onMenuItemClick()`
6. Create a method to perform an action for each popup menu item

Add an ImageButton

```
<ImageButton
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:id="@+id/button_popup"
 android:src="@drawable/@drawable/ic_action_popup"/>
```

Assign onClickListener to button

```
private ImageButton mButton =
    (ImageButton) findViewById(R.id.button_popup);
```

In onCreate:

```
mButton.setOnClickListener(new View.OnClickListener() {
    // define onClick
});
```

Implement onClick

```
@Override
public void onClick(View v) {
    PopupMenu popup = new PopupMenu(MainActivity.this, mButton);
    popup.getMenuInflater().inflate(
        R.menu.menu_popup, popup.getMenu());
    popup.setOnMenuItemClickListener(
        new PopupMenu.OnMenuItemClickListener() {
            // implement click listener
         });
    popup.show();
}
```

Implement onMenuItemClick

```
public boolean onMenuItemClick(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.option_toast:
            Toast.makeText(getApplicationContext(), "Popup Toast",
                    Toast.LENGTH_SHORT).show();
            return true;
        default:
            return false;
    }
}
```

Learn more

- Adding the App Bar
- Styles and Themes
- Menus
- Menu Resource

What's Next?

- Concept Chapter: 4.2 C Menus
- Practical: 4.2 P Using an Options Menu

END

Android Developer Fundamentals

User Interaction and Intuitive Navigation

Lesson 4

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

4.3 Screen Navigation

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Contents

- Back navigation
- Hierarchical navigation
  - Up navigation
  - Descendant navigation
  - Navigation drawer for descendant navigation
  - Lists and carousels for descendant navigation
  - Ancestral navigation
  - Lateral navigation

Two forms of navigation

Temporal or back navigation

- provided by the device's back button
- controlled by the Android system's back stack

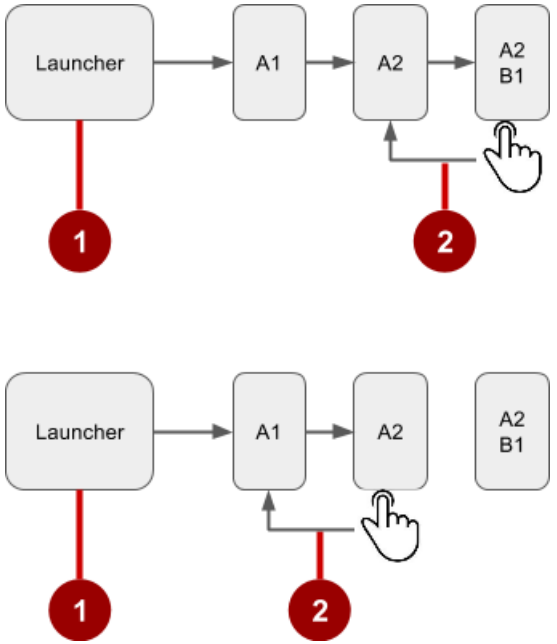
Ancestral or up navigation

- provided by the app's action bar
- controlled by defining parent-child relationships between activities in the Android manifest

Back Navigation

Navigation through history of screens

1. Historys starts from Launcher
2. User clicks the Back button to navigate to the previous screens in reverse order



Changing Back ◁ button behavior

- Android system manages the back stack and Back button
- If in doubt, don't change
- Only override, if necessary to satisfy user expectation

For example: In an embedded browser, trigger browser's default back behavior when user presses device Back button

Overriding onBackPressed()

```
@Override
public void onBackPressed() {
    // Add the Back key handler here.
    return;
}
```

Hierarchical Navigation

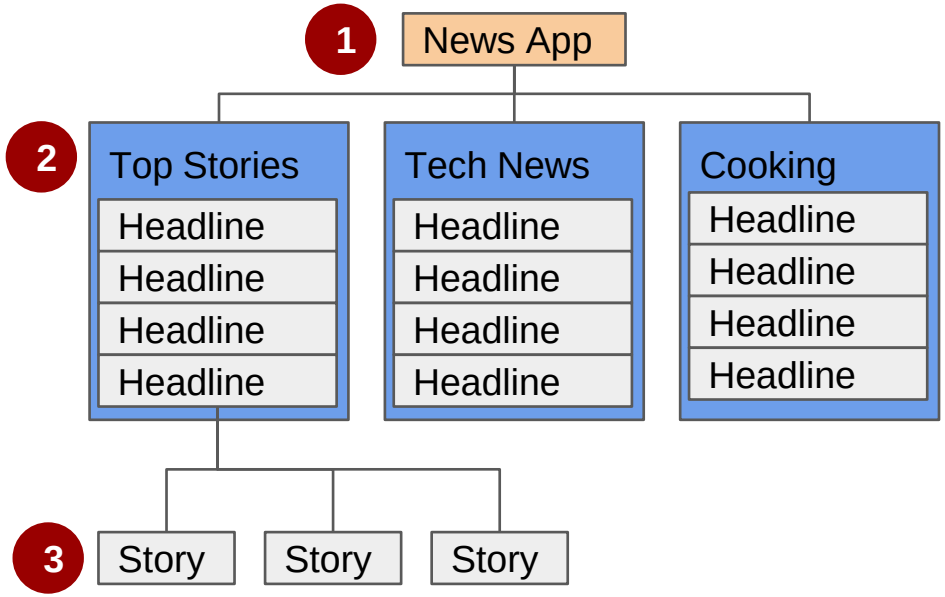
Hierarchical navigation patterns

- **Parent screen**—Screen that enables navigation down to child screens, such as home screen and main activity
- **Collection sibling**—Screen enabling navigation to a collection of child screens, such as a list of headlines
- **Section sibling**—Screen with content, such as a story

Example of a screen hierarchy

1. Parent screen
2. Children: collection siblings
3. Children: section siblings

Use activities or fragments to implement a hierarchy



Types of hierarchical navigation

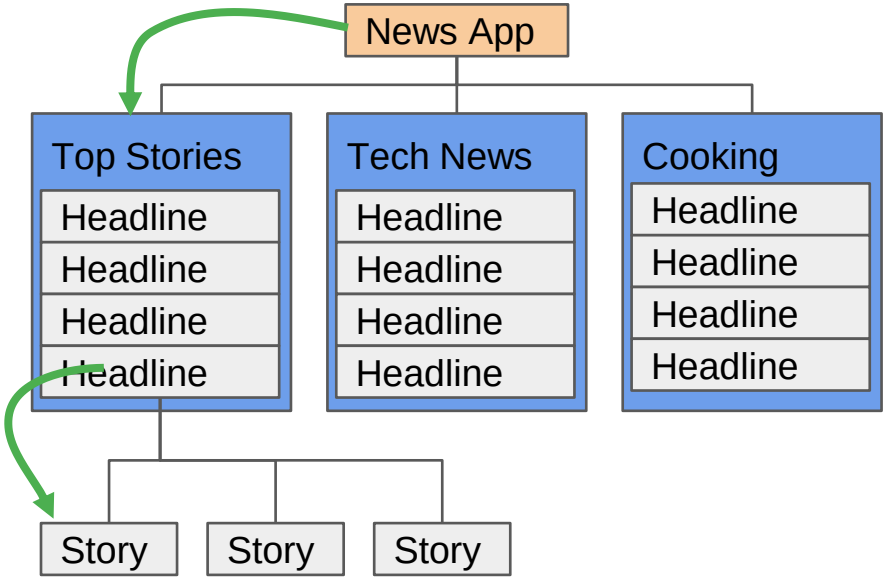
- Descendant navigation
  - Down from a parent screen to one of its children
  - From a list of headlines to a story summary to a story
- Ancestral navigation
  - Up from a child or sibling screen to its parent
  - From a story summary back to the headlines
- Lateral navigation
  - From one sibling to another sibling
  - Swiping between tabbed views

Descendant Navigation

Descendant navigation

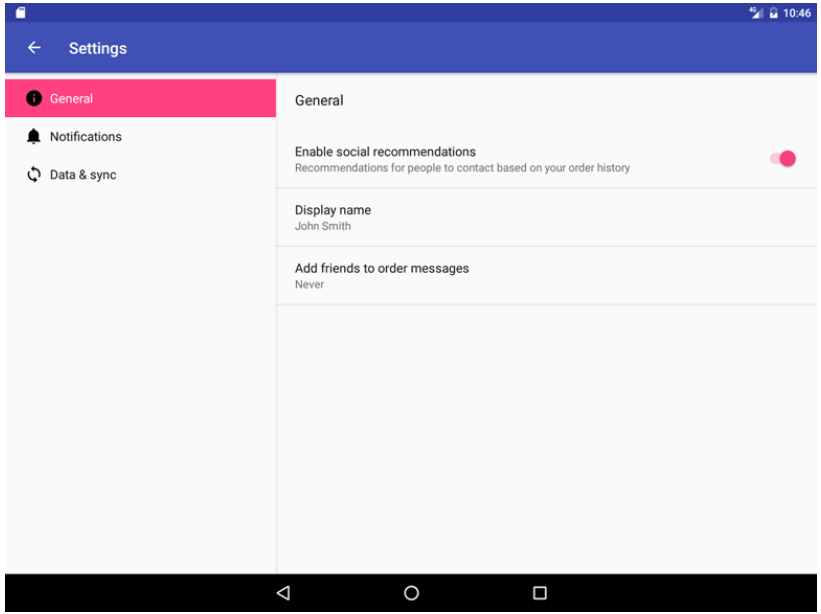
Descendant navigation

- Down from a parent screen to one of its children
- From the main screen to a list of headlines to a story

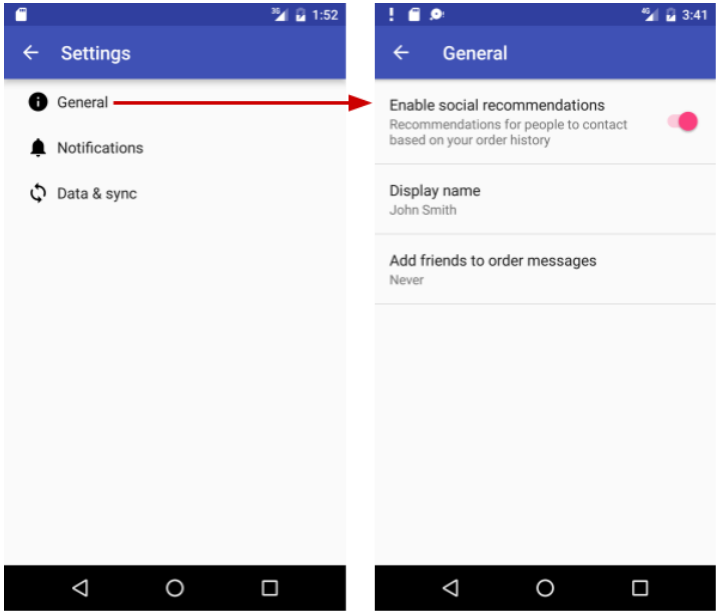


Master/detail flow

- Side-by side on tablets



- Multiple screens on phone



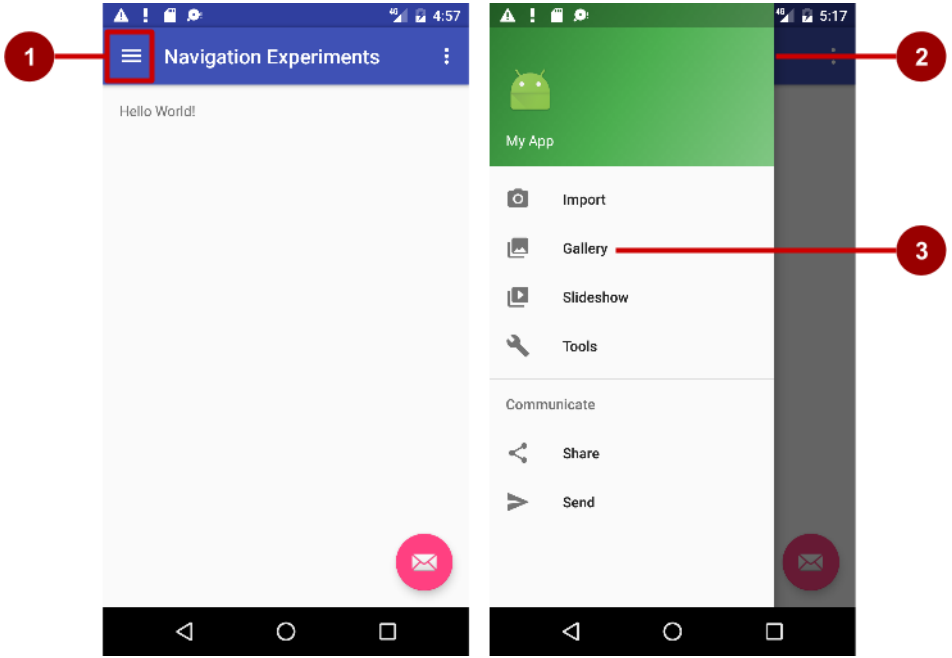
Controls for descendant navigation

- Buttons, image buttons on main screen
- Other clickable views with text and icons
- Arranged in horizontal or vertical rows, or as a grid
- List items on collection screens

Navigation Drawer for Descendant Navigation

Navigation drawer

1. Icon in app bar
2. Header
3. Menu items

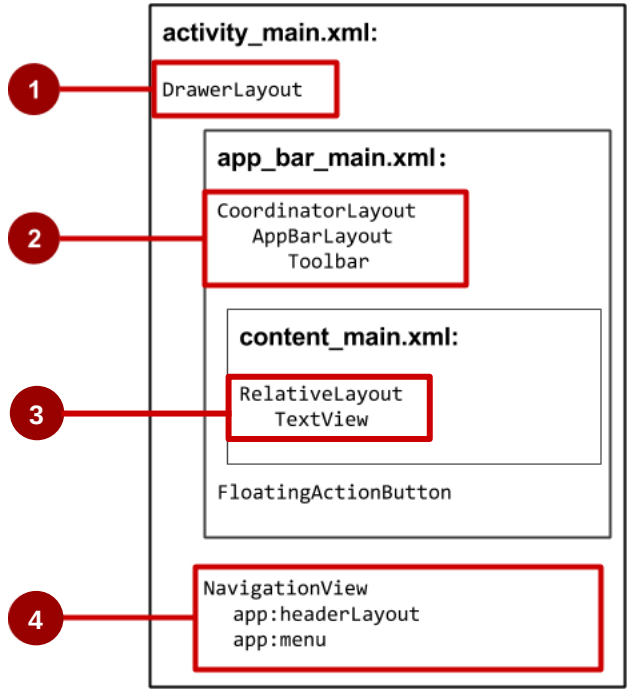


Steps for navigation drawer

1. Create layouts for drawer, drawer header, drawer menu items, app bar, activity screen contents
2. Add navigation drawer and item listeners to activity code
3. Handle the navigation drawer menu item selections

Navigation drawer activity layout

1. DrawerLayout is root view
2. CoordinatorLayout contains app bar layout with a Toolbar
3. App content screen layout
4. NavigationView with layouts for header and selectable items



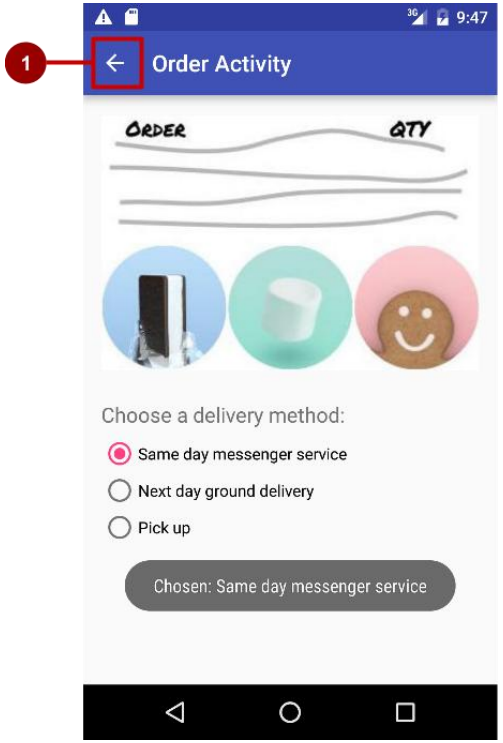
Other descendant navigation patterns

- Vertical list, such as RecyclerView
- Vertical grid, such as GridView
- Lateral navigation with a Carousel
- Multi-level menus, such as the Options menu
- Master/detail navigation flow

Ancestral Navigation

Ancestral navigation (Up button)

Enable user to go up from a section or child screen to the parent



Declare activity's parent in Android manifest

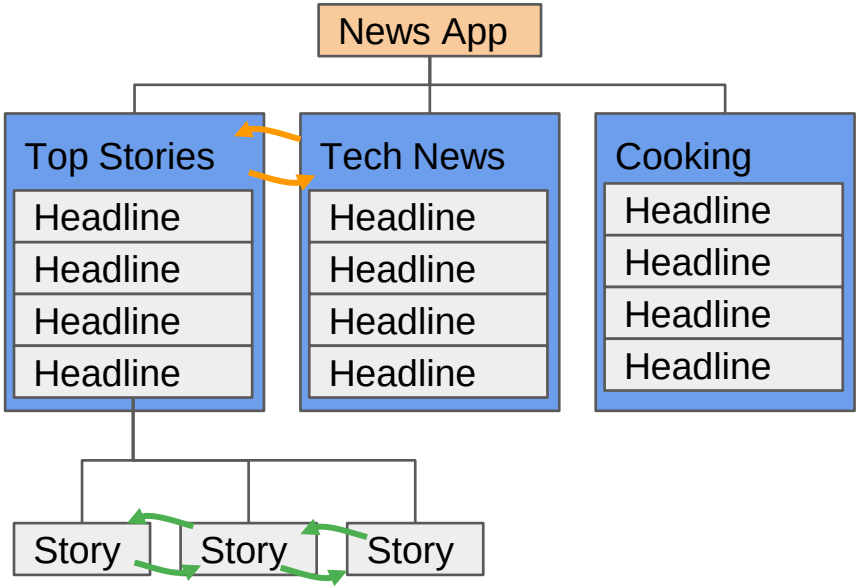
```
<activity android:name=".OrderActivity"
  android:label="@string/title_activity_order"
  android:parentActivityName="com.example.android.
                        optionsmenuorderactivity.MainActivity">
  <meta-data
     android:name="android.support.PARENT_ACTIVITY"
     android:value=".MainActivity"/>
</activity>
```

Lateral Navigation

Tabs and swipes

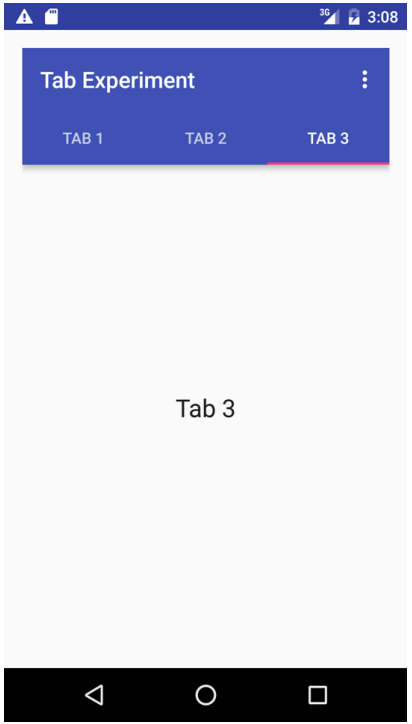
Lateral navigation

- Between siblings
- From a list of stories to a list in a different tab
- From story to story under the same tab



Benefits of using tabs and swipes

- A single, initially-selected tab—users have access to content without further navigation
- Navigate between related screens without visiting parent



Best practices with tabs

- Lay out horizontally
- Run along top of screen
- Persistent across related screens
- Switching should not be treated as history

Steps for implementing tabs

1. Define the tab layout using TabLayout
2. Implement a fragment and its layout for each tab
3. Implement a PagerAdapter from FragmentPagerAdapter or FragmentStatePagerAdapter
4. Create an instance of the tab layout
5. Manage screen views in fragments
6. Set a listener to determine which tab is tapped

See Practical for coding details; summary in following slides

Add tab layout below Toolbar

```
<android.support.design.widget.TabLayout
    android:id="@+id/tab_layout"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@id/toolbar"
    android:background="?attr/colorPrimary"
    android:minHeight="?attr/actionBarSize"
    android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"/>
```

Add view pager below TabLayout

```
<android.support.v4.view.ViewPager
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="fill_parent"
    android:layout_below="@id/tab_layout" />
```

Create a tab layout in onCreate()

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.tab_layout);
tabLayout.addTab(tabLayout.newTab().setText("Tab 1"));
tabLayout.addTab(tabLayout.newTab().setText("Tab 2"));
tabLayout.addTab(tabLayout.newTab().setText("Tab 3"));
tabLayout.setTabGravity(TabLayout.GRAVITY_FILL);
```

Add the view pager in onCreate()

```
final ViewPager viewPager = (ViewPager) findViewById(R.id.pager);
final PagerAdapter adapter = new PagerAdapter (
    getSupportFragmentManager(), tabLayout.getTabCount());
viewPager.setAdapter(adapter);
```

Add the listener in onCreate()

```
viewPager.addOnPageChangeListener(
            new TabLayout.TabLayoutOnPageChangeListener(tabLayout));
tabLayout.addOnTabSelectedListener(
            new TabLayout.OnTabSelectedListener() {
   @Override
   public void onTabSelected(TabLayout.Tab tab) {
       viewPager.setCurrentItem(tab.getPosition());}
   @Override
   public void onTabUnselected(TabLayout.Tab tab) {}
   @Override
   public void onTabReselected(TabLayout.Tab tab) {} });
```

Learn more

- Navigation Design guide
  d.android.com/design/patterns/navigation.html
- Designing effective navigation
  d.android.com/training/design-navigation/index.html
- Creating a Navigation Drawer
  d.android.com/training/implementing-navigation/nav-drawer.html
- Creating swipe views with tabs
  d.android.com/training/implementing-navigation/lateral.html

What's Next?

- Concept Chapter: 4.3 C Screen Navigation
- Practical: 4.3 P Using the App Bar and Tabs for Navigation

END

Android Developer Fundamentals

User Interaction and Intuitive Navigation

Lesson 4

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

4.4 Recycler View

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Contents

- RecyclerView Components
- Implementing a RecyclerView

What is a Recycler View?

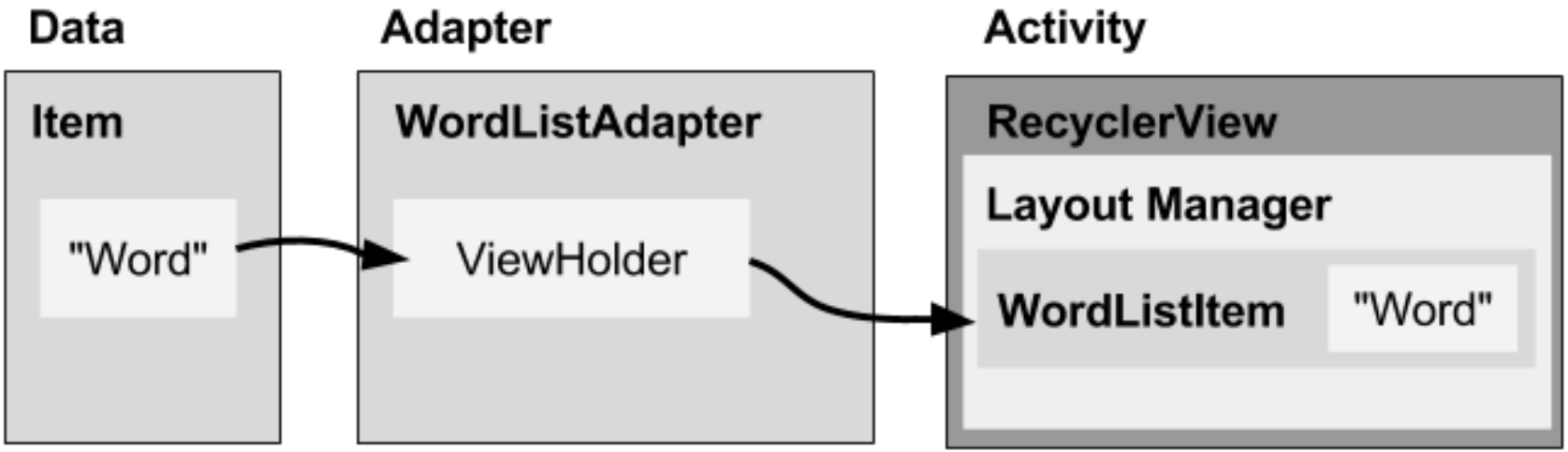
- Scrollable container for large data sets
- Efficient
  - uses and reuses limited number of views
  - Updates changing data fast
- RecyclerView

RecyclerView Components

All Components overview

- **Data**
- **RecyclerView** scrolling list for list items—RecyclerView
- **Layout** for one item of data—XML file
- **Layout manage**r handles the organization of UI components in a view—Recyclerview.LayoutManager
- **Adapter** connects data to the RecyclerView—RecyclerView.Adapter
- **View holder** has view information for displaying one item—RecyclerView.ViewHolder

How components fit together overview

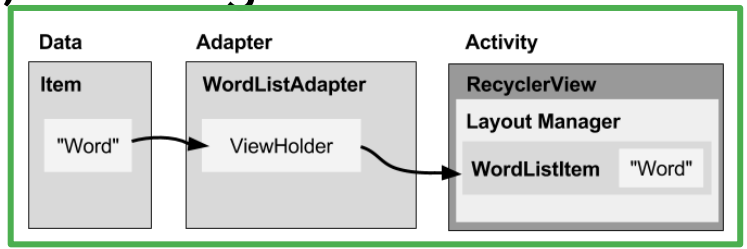


What is a layout manager?

- All view groups have layout managers
- Positions item views inside a RecyclerView.
- Reuses item views that are no longer visible to the user
- Built-in layout managers include LinearLayoutManager, GridLayoutManager, and StaggeredGridLayoutManager
- For RecyclerView, extend RecyclerView.LayoutManager

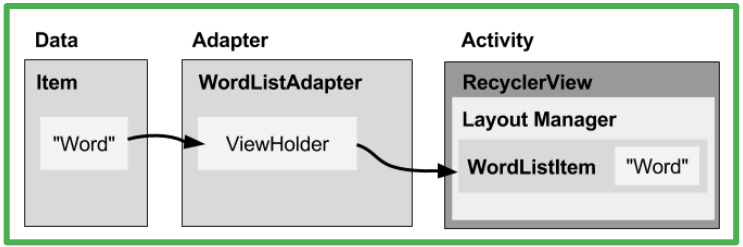
What is an adapter?

- Helps incompatible interfaces work together, for example, takes data from a database Cursor and puts them as strings into a view
- Intermediary between data and view
- Manages creating, updating, adding, deleting item views as the underlying data changes
- RecyclerView.Adapter



What is a view holder?

- Used by the adapter to prepare one view with data for one list item
- Layout specified in an XML resource file
- Can have clickable elements
- Is placed by the layout manager
- RecyclerView.ViewHolder



Implementing RecyclerView

Steps Summary

1. Add the RecyclerView dependency to app/build.gradle file
2. Add RecyclerView to layout
3. Create XML layout for item
4. Extend RecyclerView.Adapter
5. Extend RecyclerView.ViewHolder
6. In onCreate of activity, create a RecyclerView with adapter and layout manager

Add dependency to app/build.gradle

```
dependencies {
   ...
   compile 'com.android.support:recyclerview-v7:24.1.1'
   ...
   }
```

Add RecyclerView to XML Layout

```
<android.support.v7.widget.RecyclerView
    android:id="@+id/recyclerview"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
</android.support.v7.widget.RecyclerView>
```

Create layout for 1 list item

```
<LinearLayout …>
   <TextView
      android:id="@+id/word"
      style="@style/word_title" />
</LinearLayout>
```

Implement the adapter

```
public class WordListAdapter
    extends RecyclerView.Adapter<WordListAdapter.WordViewHolder> {

    public WordListAdapter(Context context,
                           LinkedList<String> wordList) {
        mInflater = LayoutInflater.from(context);
        this.mWordList = wordList;
    }
}
```

Adapter has 3 required methods

- `onCreateViewHolder()`
- `inBindViewHolder()`
- `getItemCount()`

Let's take a look!

onCreateViewHolder()

```
@Override
    public WordViewHolder onCreateViewHolder(
        ViewGroup parent, int viewType) {
        // Create view from layout
        View mItemView = mInflater.inflate(
             R.layout.wordlist_item, parent, false);
        return new WordViewHolder(mItemView, this);
     }
```

onBindViewHolder()

```
@Override
    public void onBindViewHolder(
        WordViewHolder holder, int position) {
        // Retrieve the data for that position
        String mCurrent = mWordList.get(position);
        // Add the data to the view
        holder.wordItemView.setText(mCurrent);
    }
```

getItemCount()

```
@Override
public int getItemCount() {
   // Return the number of data items to display
   return mWordList.size();
}
```

Create the view holder in adapter class

```
class WordViewHolder extends RecyclerView.ViewHolder {}
```

If you want to handle mouse clicks:

```
class WordViewHolder extends RecyclerView.ViewHolder
                     implements View.OnClickListener {}
```

View holder constructor

```
public WordViewHolder(View itemView, WordListAdapter adapter) {
  super(itemView);
  // Get the layout
  wordItemView = (TextView) itemView.findViewById(R.id.word);
  // Associate with this adapter
  this.mAdapter = adapter;
  // Add click listener, if desired
  itemView.setOnClickListener(this);
}
// Implement onClick() if desired
```

Create the RecyclerView in activity's onCreate()

```
mRecyclerView = (RecyclerView)
findViewById(R.id.recyclerview);
mAdapter = new WordListAdapter(this, mWordList);
mRecyclerView.setAdapter(mAdapter);
mRecyclerView.setLayoutManager(new
LinearLayoutManager(this));
```

Practical: RecyclerView

- This is rather complex with many separate pieces. So, there is a whole practical where you implement a RecyclerView that displays a list of clickable words.

- Shows all the steps, one by one with a complete app

Learn more

- RecyclerView
- RecyclerView class
- RecyclerView.Adapter class
- RecyclerView.ViewHolder class
- RecyclerView.LayoutManager class

What's Next?

- Concept Chapter: 4.4 C RecyclerView
- Practical: 4.4 P Create a Recycler View

END

Android Developer Fundamentals

Delightful User Experience

Lesson 5

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

5.1 Drawables, Styles, Themes

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Contents

- Drawables
- Creating image assets
- Styles
- Themes

Drawables

Drawables

- Drawable—generic Android class used to represent any kind of graphic
- All drawables are stored in the **res/drawable** project folder

Drawable classes

Bitmap File
Nine-Patch File
Layer List Drawable
Shape Drawable
State List Drawable
Level List Drawable

Transition Drawable
Vector Drawable
… and more
Custom Drawables

Bitmaps

- PNG (.png), JPG (.jpg), or GIF (.gif) format
- Uncompressed BMP (.bmp)
- WebP (4.0 and higher)
- Creates a BitmapDrawable data type
- Placed directly in res/drawables

Referencing Drawables

- XML: `@[package:]drawable/filename`

```
<ImageView
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:src="@drawable/myimage" />
```

- Java code: `R.drawable.filename`

```
Resources res = getResources();
Drawable drawable = res.getDrawable(R.drawable.myimage);
```

Nine-Patch Files

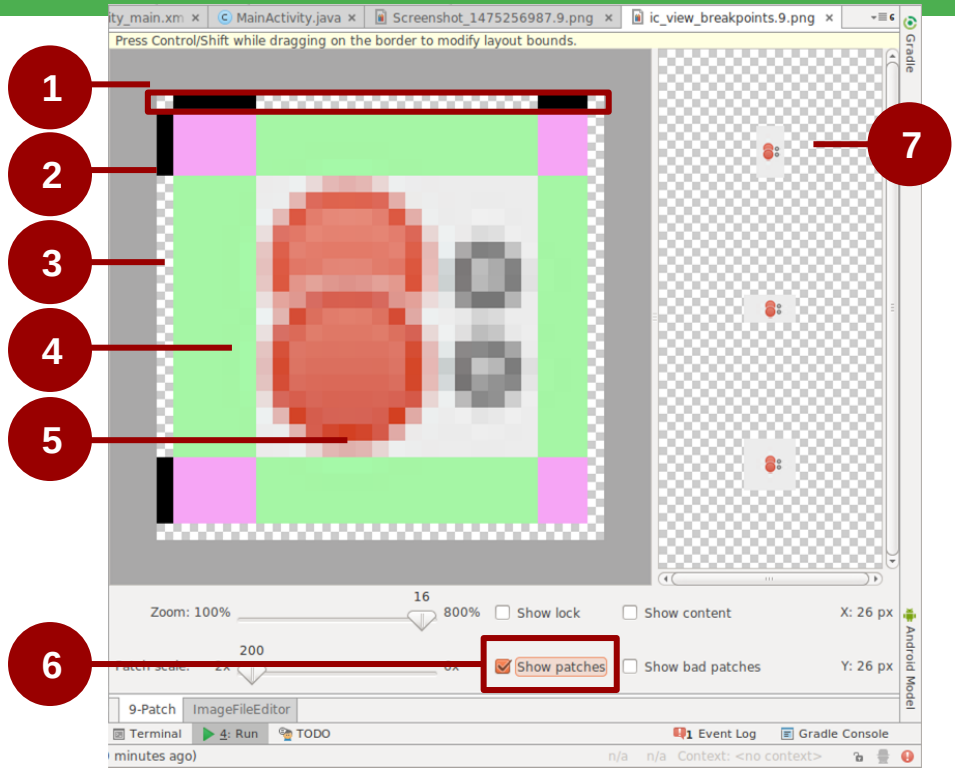
- Nine-patch files (.9.png) are PNG with stretchable regions
- Only stretches bigger, not smaller, so start with small image
- Often used for backgrounds of UI elements
- Example: button background changes size with label length
- Good intro

Creating Nine-Patch Files

1. Put a small PNG file into **res/drawable**
2. Right-click and choose **Create 9-Patch file**
3. **Double-click** 9-Patch file to open editor
4. Specify the stretchable regions (next slide)

Editing Nine-Patch Files

1. Border to mark stretchable regions for width
2. Stretchable regions marked for height
   Pink == both directions
3. Click to turn pixels black. Shift-click (ctrl-click on Mac) to unmark
4. Stretchable area
5. Not stretchable
6. Check **Show patches**
7. Preview of stretched image

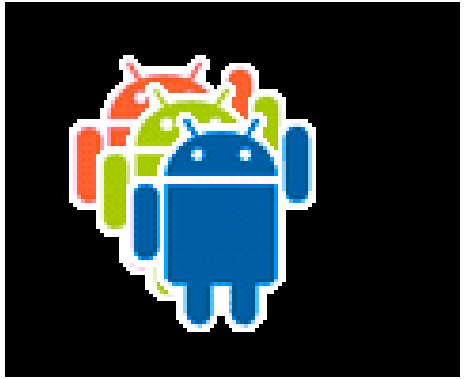


Layer List

- You can create layered images, just like with drawing tools, such as Gimp
- In Android, each layer is represented by a drawable
- Layers are organized and managed in XML
- List and the items can have properties
- Layers are drawn on top of each other in the order defined in the XML file
- LayerDrawable

Creating Layer List

```
<layer-list>
    <item>
      <bitmap android:src="@drawable/android_red"
        android:gravity="center" />
    </item>
    <item android:top="10dp" android:left="10dp">
      <bitmap android:src="@drawable/android_green"
        android:gravity="center" />
    </item>
    <item android:top="20dp" android:left="20dp">
      <bitmap android:src="@drawable/android_blue"
        android:gravity="center" />
    </item>
</layer-list>
```



Shape Drawables & GradientDrawable

- Define a shape and its properties in XML
  - Rectangle, oval, ring, line
- Styled with attributes such as <corners>, <gradient>, <padding>, <size>, <solid> and <stroke>

  See Shape Drawables for more attributes
- Can be inflated for a GradientDrawable

Creating a GradientDrawable

```
<shape … android:shape="rectangle">
  <gradient
     android:startColor="@color/white"
     android:endColor="@color/blue"
     android:angle="45"/>
  <corners android:radius="8dp" />
</shape>
```


```
Resources res = getResources();
Drawable shape = res.getDrawable(R.drawable.gradient_box);

TextView tv = (TextView)findViewByID(R.id.textview);
tv.setBackground(shape);
```

Transition Drawables

- Drawable that can cross-fade between two other drawables
- Each graphic represented by <item> inside <selector>
- Represented by TransitionDrawable in Java code
- Transition forward by calling `startTransition()`
- Transition backward with `reverseTransition()`

Creating Transition Drawables

```
<transition ...>
  <selector> <item android:drawable="@drawable/on" />
             <item android:drawable="@drawable/off" />
  </selector>
</transition>
```

```
<ImageButton
    android:id="@+id/button"
    android:src="@drawable/transition" />
```

```
ImageButton button = (ImageButton) findViewById(R.id.button);
TransitionDrawable drawable =
        (TransitionDrawable) button.getDrawable();
drawable.startTransition(500);
```

Vector drawables

- Scale smoothly for all screen sizes
- Android API Level 21 and up
- Use Vector Asset Studio to create (slides below)
- VectorDrawable

Creating Vector drawables

```
<vector ...
    android:height="256dp" android:width="256dp"
    android:viewportWidth="32" android:viewportHeight="32">
  <path android:fillColor="@color/red"
      android:pathData="M20.5,9.5
                        c-1.955,0,-3.83,1.268,-4.5,3
                        c-0.67,-1.732,-2.547,-3,-4.5,-3 ... />
</vector>
```

pathData for heart shape

Image Asset Studio

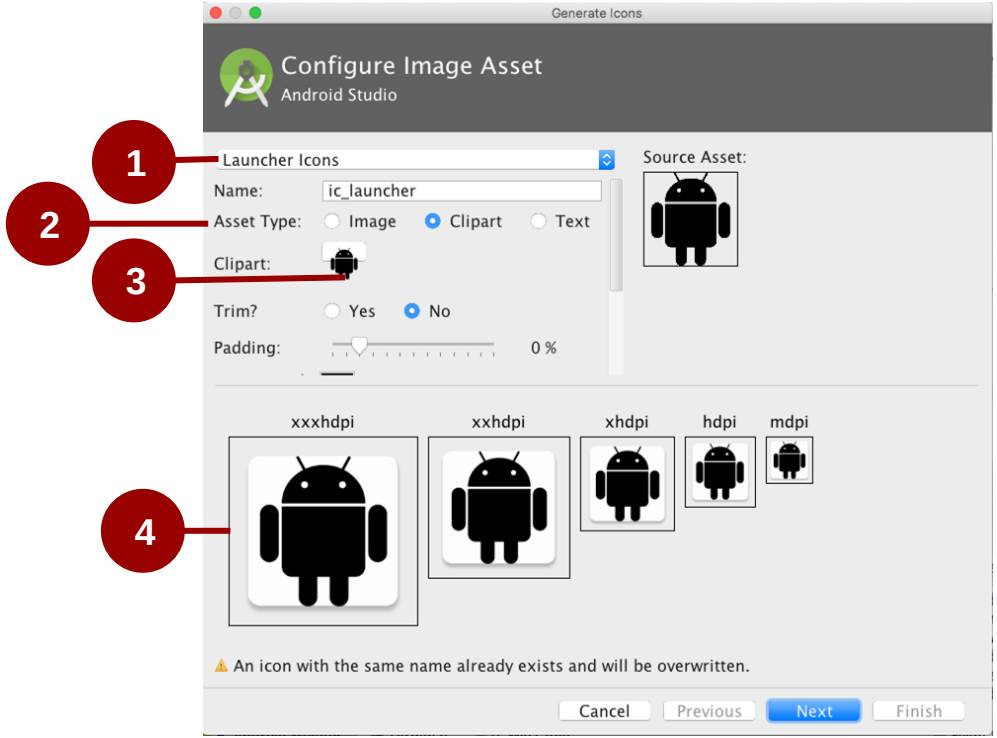
What is Image Asset Studio?

- Create icons from material icons, images, and text
- Launcher, action bar, tab, notification icons
- Generates a set of icons for generalized screen density
- Stored in **/res** folder

- To start Image Asset Studio
  1. Right-click **res** folder of your project
  2. Choose **New > Image Asset**

Using Image Asset Studio

1. Chose icon type and change name
2. Choose Image, Clipart, or Text
3. Click icon to chose clipart
4. Inspect assets for multiple screen sizes



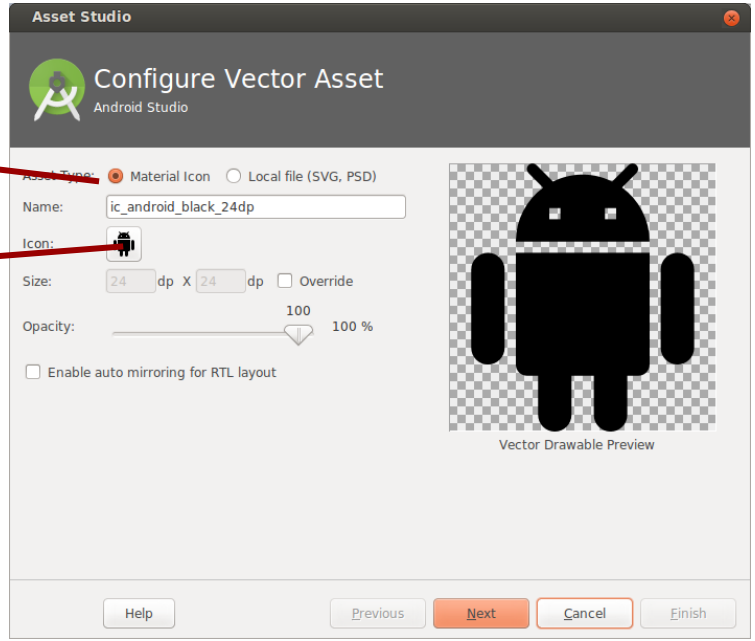
Vector Asset Studio

What is Vector Asset Studio?

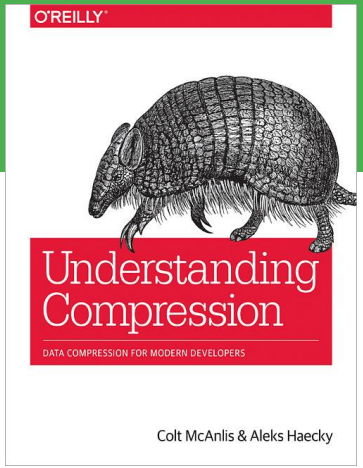
- Create icons from material icons or supply your own vector drawings for API 21 and later
- Launcher, action bar, tab, notification icons
- Generates a scalable vector drawable
- Stored in **/res** folder
- To start Image Asset Studio
  1. Right-click **res** folder of your project
  2. Choose **New > Vector Asset**

Using Image Asset Studio

1. Choose from Material Icon library, or supply your own SVG or PSD vector drawing
2. Opens Material Icon library



Images, memory, and performance



- Use smallest resolution picture necessary
- Resize, crop, compress
- Vector drawings for simple images
- Use Libraries: Glide or Picasso
- Choose appropriate image formats for image type and size
- Use lossy image formats and adjust quality where possible
- Learn about data compression for developers from Understanding Compression

Styles

What is a Style?

- Collection of attributes that define the visual appearance of a View
- Reduce duplication
- Make code more compact
- Manage visual appearance of many components with one style

Styles reduce clutter

```
<TextView
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:textColor="#00FF00"
  android:typeface="monospace"
  android:text="@string/hello" />
```

```
<TextView
    style="@style/CodeFont"
    android:text="@string/hello"
/>
```

Define styles in styles.xml

styles.xml is in **/res/values**

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="CodeFont">
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
</resources>
```

Inheritance: Parent

Define a parent style...

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="CodeFont">
        <item name="android:layout_width">match_parent</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
</resources>
```

Inheritance: Define child

Define child with Codefont as parent

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="RedCode" parent="@style/Codefont>
        <item name="android:textColor">#FF0000</item>
    </style>
</resources>
```

Themes

Themes

- A Theme is a style applied to an entire activity or even the entire application
- Themes are applied in the Android Manifest

```
<application android:theme="@style/AppTheme">
```

Customize AppTheme of Your Project

```
<!-- Base application theme. -->
<style name="AppTheme"
       parent="Theme.AppCompat.Light.DarkActionBar">
<!-- Try: Theme.AppCompat.Light.NoActionBar -->
   <!-- Customize your theme here. -->
   <item name="colorPrimary">@color/colorPrimary</item>
   <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
   <item name="colorAccent">@color/colorAccent</item>
</style>
```

Styles and Themes Resources

Android platform has collection of built in styles and themes

- Android Styles
- Android Themes
- Styles and Themes Guide
- DayNight Theme Guide

Learn more

- Drawable Resource Documentation
- ShapeDrawable
- LinearLayout Guide
- Drawable Resource Guide
- Supported Media formats
- 9-Patch
- Understanding Compression

What's Next?

- Concept Chapter: 5.1 C Drawables, Styles, and Themes
- Practical: 5.1 P Drawables, Styles, and Themes

END

Delightful User Experience

Android Developer Fundamentals

Lesson 5

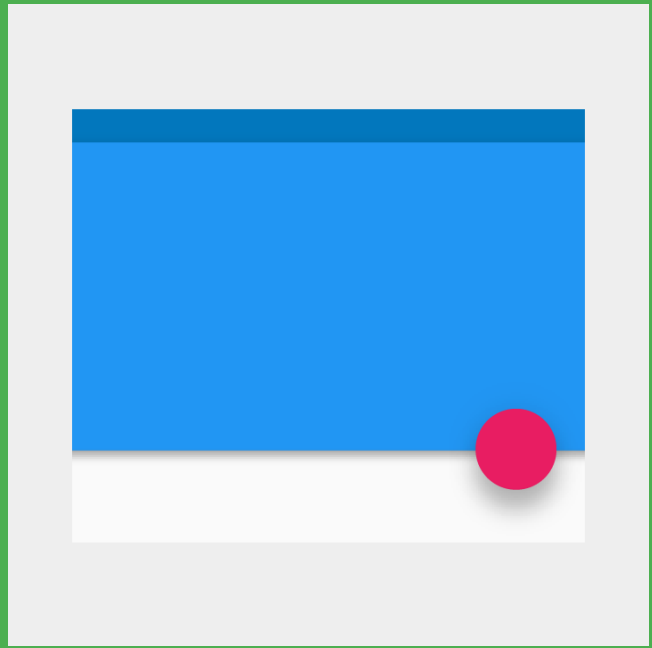
This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

5.2 Material Design

Contents

- The Material Metaphor
- Imagery
- Typography
- Color
- Motion
- Layout
- Components

The Material Metaphor



What is Material Design?

- Design guidelines
- Visual language
- Combine classic principles of good design with innovation and possibilities of technology and science
- Material Design Spec

Material metaphor

- Three-dimensional environment containing light, material, and shadows
- Surfaces and edges provide visual cues grounded in reality
- Fundamentals of light, surface, and movement convey how objects move, interact, and exist in space and in relation to each other

Bold, graphic, intentional

- Choose colors deliberately
- Fill screen edge to edge
- Use large-scale typography
- Use white space intentionally
- Emphasize user action
- Make functionality obvious

Imagery

Imagery

Images help you communicate and differentiate your app

Should be

- Relevant
- Informative
- Delightful

Best practices

- Use together with text
- Original images
- Provide point of focus
- Build a narrative

Typography


Quantum Mechanics
REGULAR
6.626069 x 10 - 34
THIN
One hundred percent cotton bond
BOLD ITALIC
Quasiparticles
BOLD
It became the non-relativistic limit of quantum field theory
CONDENSED
PAPERCRAFT
LIGHT ITALIC
Probabilistic wave - particle wavefunction orbital path
MEDIUM ITALIC
ENTANGLED
BLACK
Cardstock 80lb ultra-bright orange
MEDIUM
STATIONERY
THIN
POSITION, MOMENTUM & SPIN
CONDENSED LIGHT

Roboto typeface

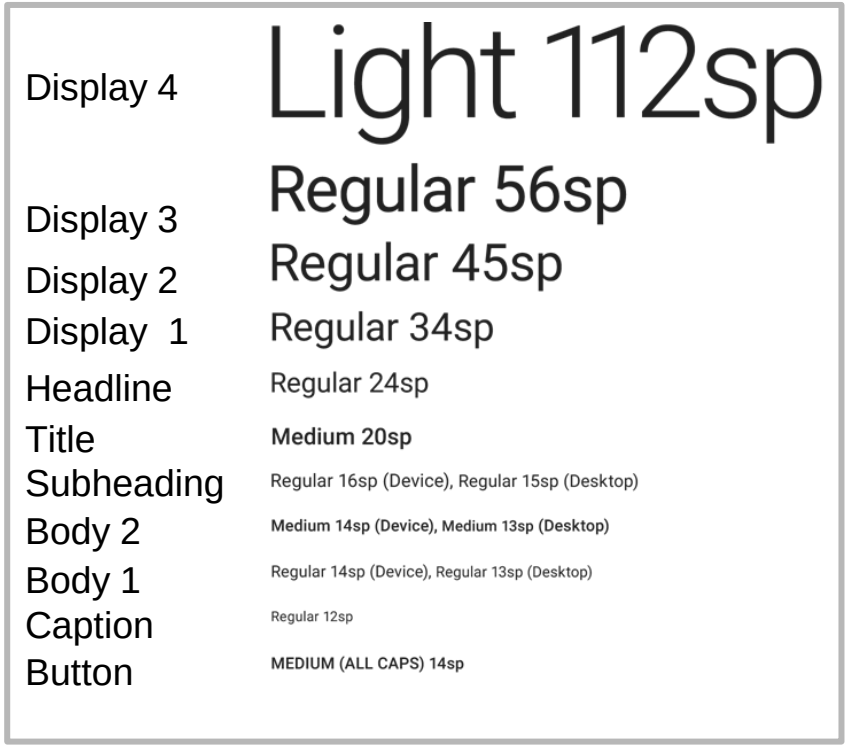
Roboto is the standard typeface on Android

Roboto has 6 weights

- Thin
- Light
- Regular
- Medium
- Bold
- Black

Font styles and scale

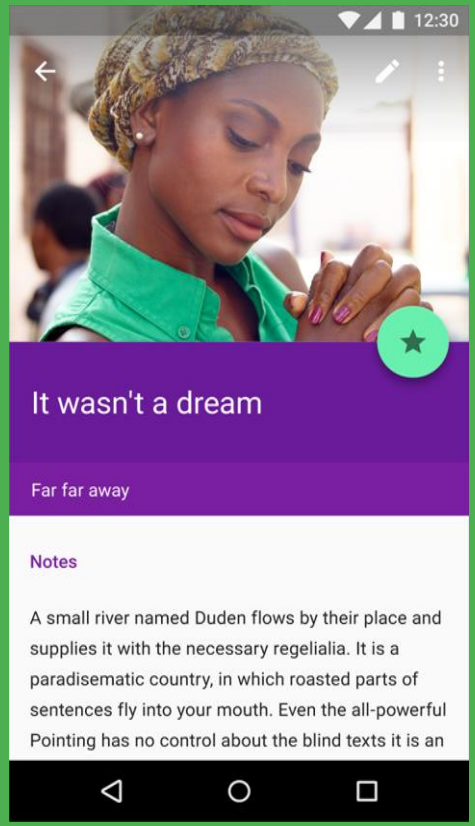
- Too many sizes is confusing and looks bad
- Limited set of sizes that work well together



Setting text appearance

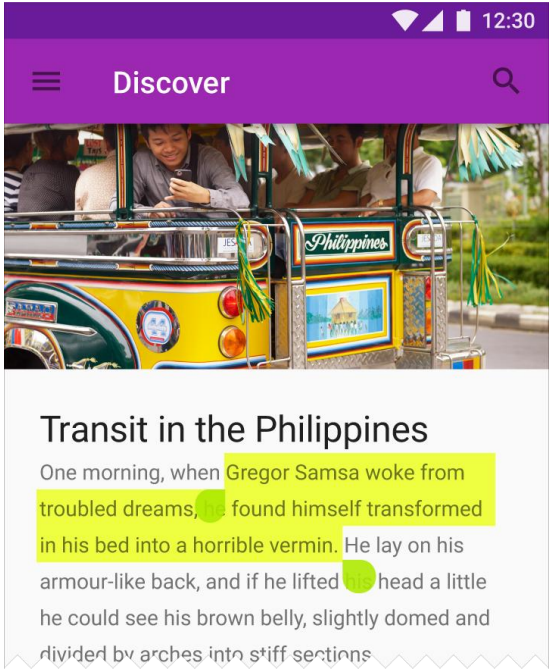
```
android:textAppearance=
    "@style/TextAppearance.AppCompat.Display3"
```

Color


12:30
It wasn't a dream
Far far away
Notes
A small river named Duden flows by their place and supplies it with the necessary regelialia. It is a paradisematic country, in which roasted parts of sentences fly into your mouth. Even the all-powerful Pointing has no control about the blind texts it is an

Color

- Bold hues
- Muted environments
- Deep shadows
- Bright highlights



Color palette

Material Design recommends using

- a primary color
- along with some shades
- and an accent color

Create a bold user experience for your app

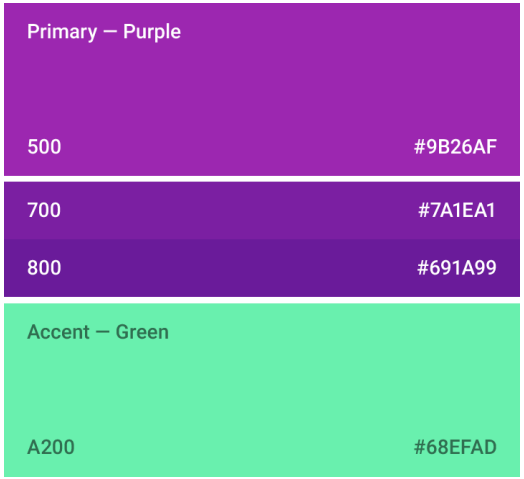
Uses of accent color

Accent color is also known as Secondary color.

- It is used for:
- Buttons
- Floating action buttons
- Selection controls, like sliders and switches
- Highlighting selected text
- Progress bars
- Links and headlines

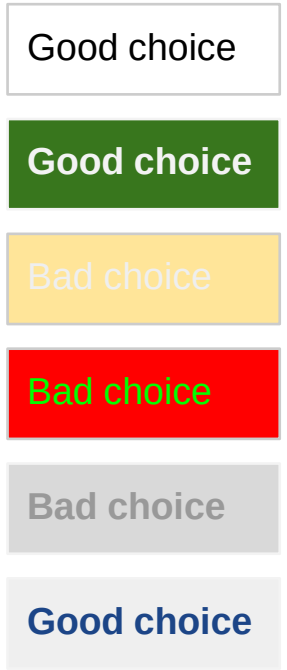
Color palette for your project

- Android Studio creates a color palette for you
- AppTheme definition in styles.xml
  - colorPrimary—AppBar, branding
  - colorPrimaryDark—status bar, contrast
  - colorAccent—draw user attention, switches, FAB
- Colors defined in colors.xml
- Color selection tool

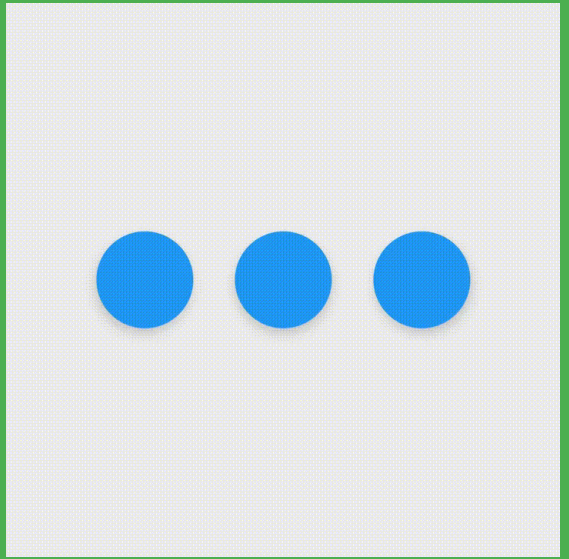


Text color and contrast

- Contrast for visual separation
- Contrast for readability
- Contrast for accessibility
- Not all people see colors the same
- Theme handles text by default
  - Theme.AppCompat.Light—text will be near black
  - Theme.AppCompat.Light.DarkActionBar—text near white



Motion



Motion

Motion in Material Design describes

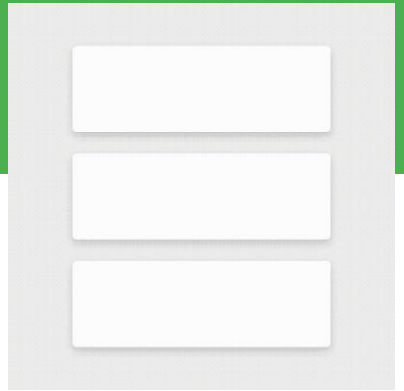
- Spatial relationships
- Functionality
- Intention

Motion is

- Responsive
- Natural
- Aware
- Intentional

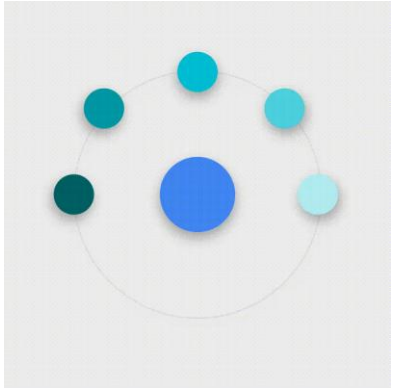
Motion in your app

- Maintain continuity
- Highlight elements or actions
- Transition naturally between actions or states
- Draw focus
- Organize transitions
- Responsive feedback

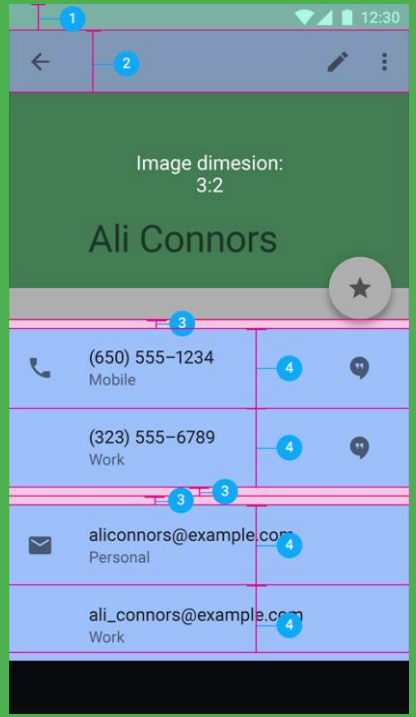


Touch feedback

Responsive interaction



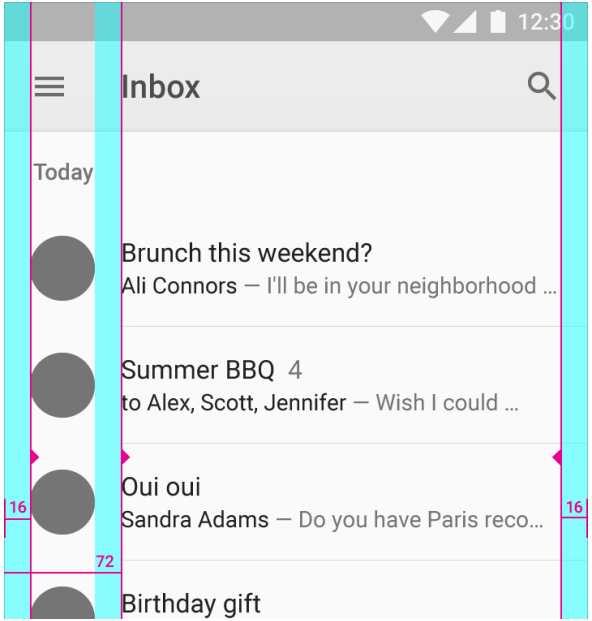
Layout


12:30
Image dimesion:
3:2
Ali Connors
(650) 555–1234
Mobile
(323) 555–6789
Work
aliconnors@example.com
Personal
ali_connors@example.com
Work

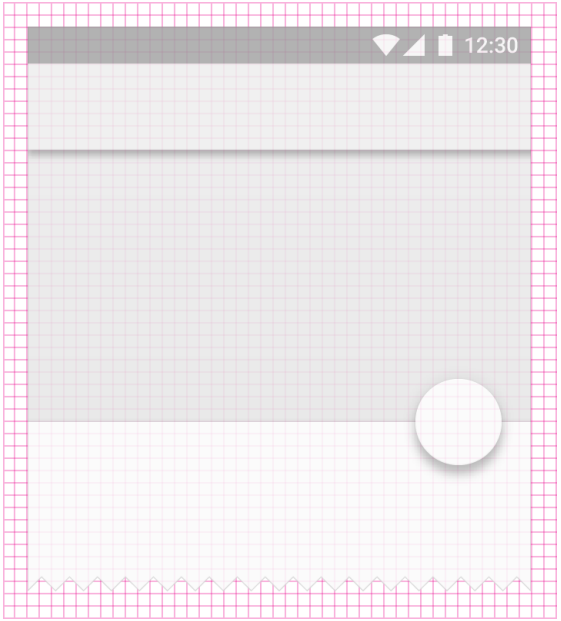
Layout for Material Design

- Density independent pixels for views—dp
- Scalable pixels for text—sp
- Elements align to a grid with consistent spacing
- Plan your layout
- Use templates for common layout patterns

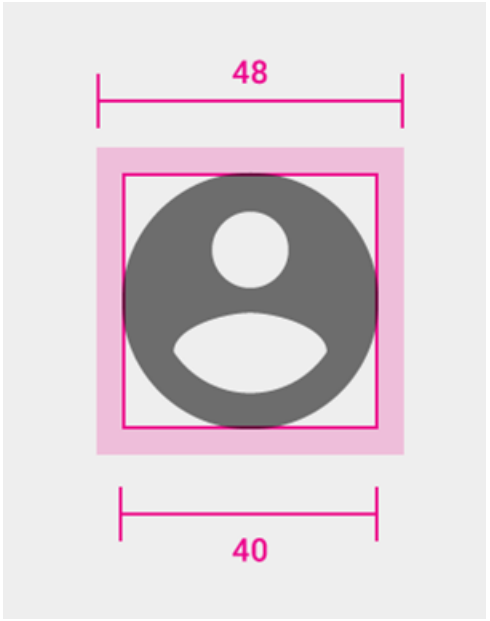
Layout planning



Spacing



Grid alignment



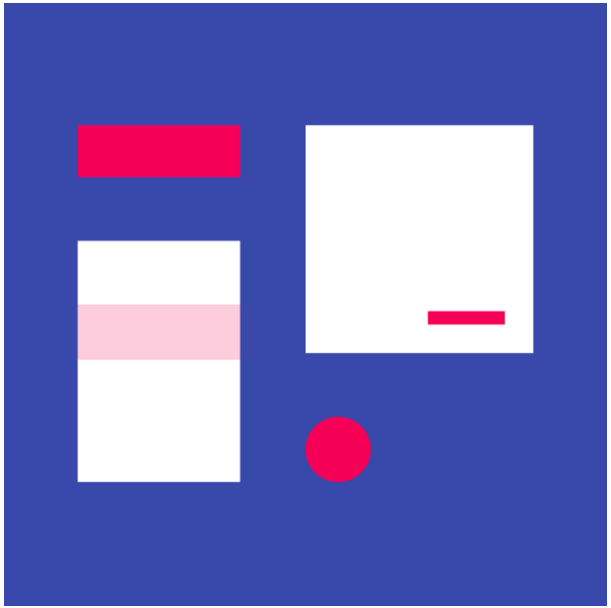
Sizing

Components

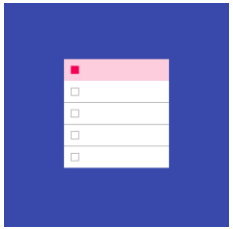
Components

Material Design has guidelines on the use and implementation of Android components

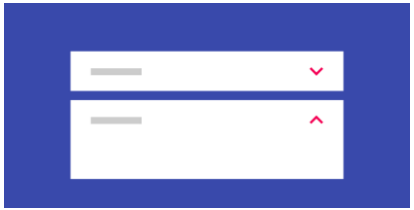
- Bottom Navigation
- Buttons
- Cards
- Chips
- Data Tables
- Dialogs
- Dividers
- Sliders
- Snackbar
- Toasts
- Steppers
- Subheaders
- Text Fields
- Toolbars



More components

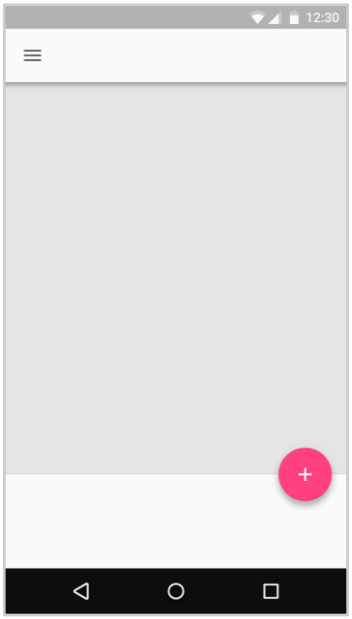


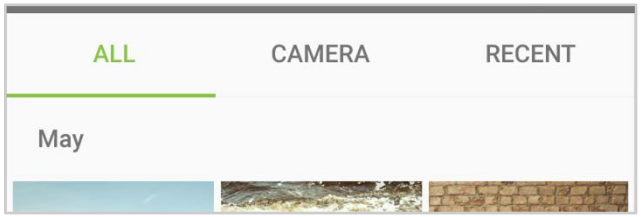
- Expansion Panels
- Grid Lists
- Lists
- Menus
- Pickers
- Progress Bars
- Selection Controls



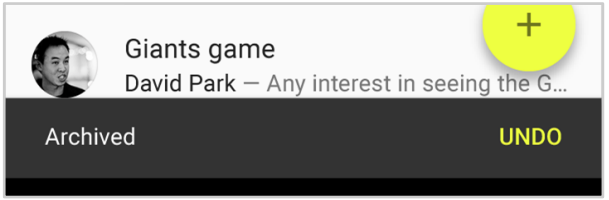
Consistency helps user intuition

FAB

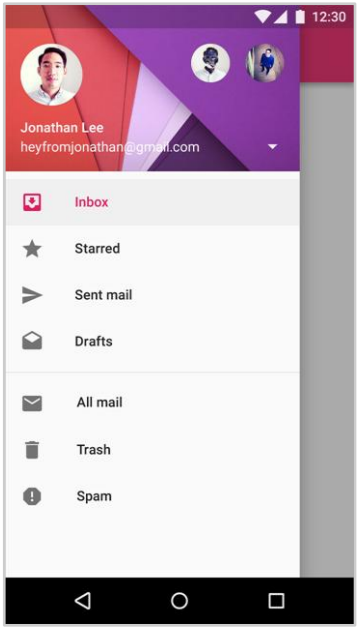




Tabs



Snackbar



Navigation Drawer

Learn more

- Material Design Guidelines
- Material Design Guide
- Material Design for Android
- Material Design for Developers
- Material Palette Generator
- Cards and Lists Guide
- Floating Action Button Reference
- Defining Custom Animations
- View Animation

What's Next?

- Concept Chapter: 5.2 C Material Design
- Practical: 5.2 P Material Design: Lists, Cards, and Colors

END

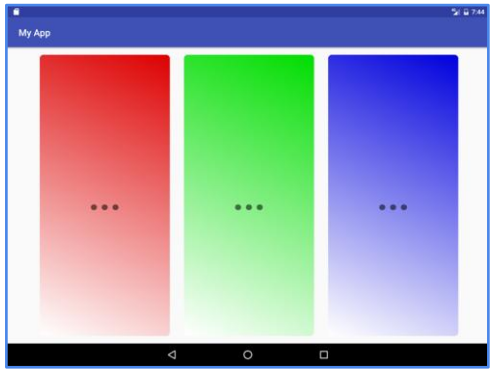
5.3 Adaptive Layouts and Resources

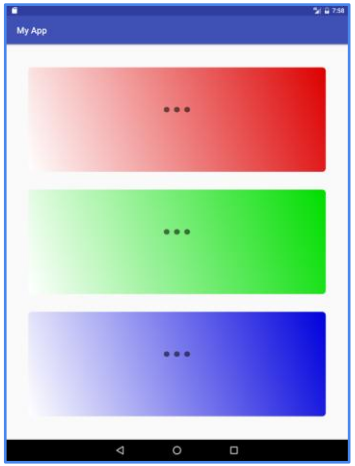
Contents

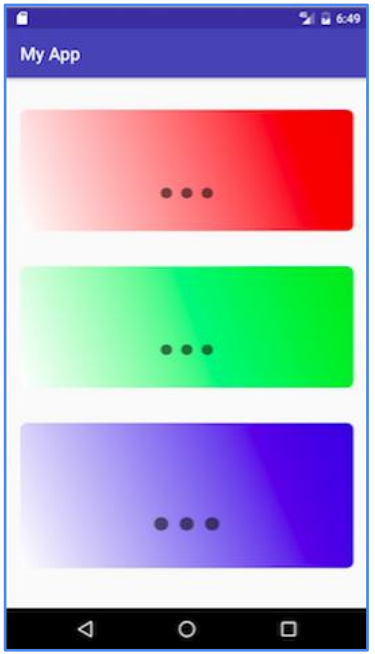
- Adaptive Layout
- Adaptive Resources
- Alternative Resources
- Default Resources

What are adaptive layouts?

Layouts that look good on different screen sizes, orientations, and devices







Adaptive Layouts and Resources

Adaptive layouts

- Layout adapts to configuration
  - Screen size
  - Device orientation
  - Locale
  - Version of Android installed
- Provides alternative resources
  - Localized strings
- Uses flexible layouts
  - GridLayout

Resource folders of a small app

```
MyProject/
    src/
    res/
        drawable/
            graphic.png
        layout/
            activity_main.xml
            list_iteminfo.xml
        mipmap/
            ic_launcher_icon.png
        values/
            strings.xml
```

Put resources in your project's res folders

Common resource directories

- `drawable/`, `layout/`, `menu/`
- `values/`—XML files of simple values, such as string or color
- `xml/`—arbitrary XML files
- `raw/`—arbitrary files in their raw form
- `mipmap/`—drawables for different launcher icon densities
- Complete list

Alternative Resources

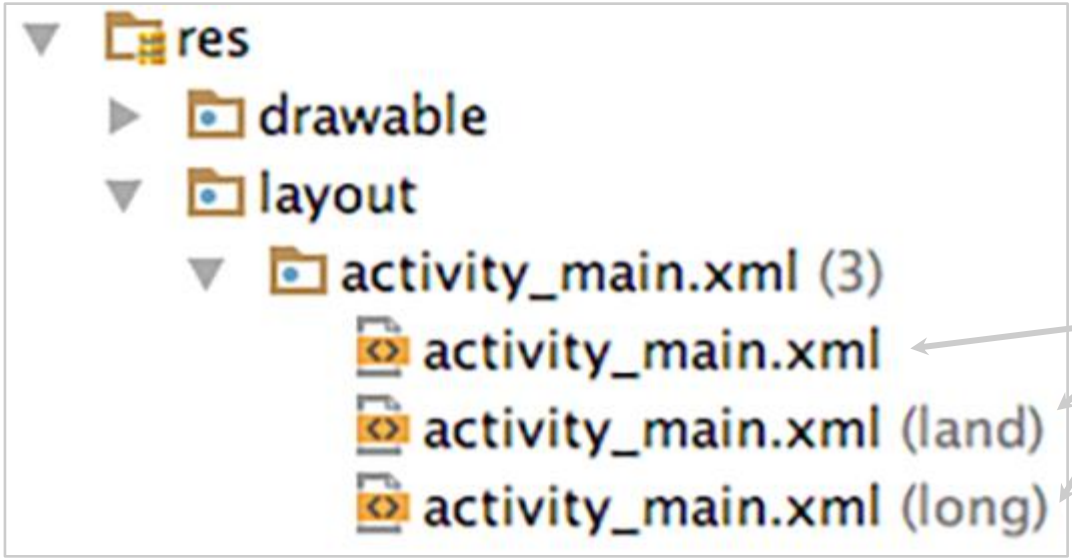
What are alternative resources?

Different device configurations may require different resources

- Localized strings
- Image resolutions
- Layout dimensions

Android loads appropriate resources automatically

Create alternative resource folders



Use alternative folders for resources for different device configurations

Names for alternative resource folders

Resource folder names have the format
`<resources name>-<config qualifier>`

`drawable-hdpi`	drawables for high-density displays
`layout-land`	layout for landscape orientation
`layout-v7`	layout for version of platform
`values-fr`	all values files for French locale

List of directories and qualifiers and usage detail

Screen Orientation

- Use `res/layout` and provide alternatives for landscape where necessary
  - `res/layout-port` for portrait-specific layouts
  - `res/layout-land` for landscape specific layouts
- Avoid hard-coded dimensions to reduce need for specialized layouts

Simple adaptive layout

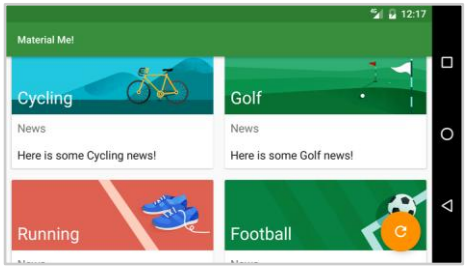
GridLayout

- In values/integer.xml:

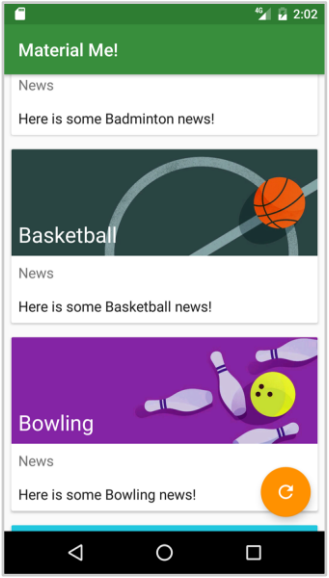
```
<integer name="grid_column_count">1</integer>
```

- In values/integer.xml-land:

```
<integer name="grid_column_count">2</integer>
```



Landscape



Portrait

Smallest width

- Smallest-width (sw) in folder name specifies minimum device width
  - res/values-sw<N>dp, where N is the smallest width
  - Example: res/values-sw600dp/dimens.xml
  - Does not change with orientation
- Android uses resource closest to (without exceeding) the device's smallest width

Platform Version

- API level supported by device
  - `res/drawables-v14`
    contains drawables for devices that support API level 14 and above
- Some resources are only available for newer versions
  - WebP image format requires API level 14 (Android 4.0)
- Android API level

Localization

- Provide strings (and other resources) for specific locales
  - `res/values-es/strings.xml`
- Increases potential audience for your app
- Locale is based on device's settings
- Localization

Default Resources

Default Resources

- Always provide default resources
  - directory name without a qualifier
  - `res/layout`, `res/values`, `res/drawables`....
- Android falls back on default resources when no specific resources match configuration
- Localizing with Resources

Learn more

- Supporting Multiple Screens
- Providing Resource
- Providing Resources Guide
- Resources Overview
- Localization Guide

What's Next?

- Concept Chapter:
  5.3 C Providing Resource for Adaptive Layouts
- Practical:
  5.3 P Supporting Landscape, Multiple Screen Sizes, and Locaction

END

Android Developer Fundamentals

Testing the User Interface

Lesson 6

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

6.1 Testing the User Interface

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Contents

- Testing Methods
- Automated Testing
- Using Espresso
- Testing Environment and Setup
- Creating Espresso Tests
- Espresso Test Examples
- Recording Tests

Testing Methods

User interface testing

- Perform all user UI actions with views
  - Tap a UI view, and enter data or make a choice
  - Examine the values of the properties of each view
- Provide input to all UI views
  - Try invalid values
- Check returned output
  - Correct or expected values?
  - Correct presentation?

Problems with testing manually

- Time consuming, tedious, error-prone
- UI may change and need frequent retesting
- Some paths fail over time
- As app gets more complex, possible sequences of actions may grow non-linearly

Benefits of testing automatically

- Free your time and resources for other work
- Faster than manual testing
- Repeatable
- Run tests for different device states and configurations

Espresso for single app testing

- Verify that the UI behaves as expected
- Check that the app returns the correct UI output in response to user interactions
- Navigation and controls behave correctly
- App responds correctly to mocked-out dependencies

UI Automator for multiple apps

- Verify that interactions between different user apps and system apps behave as expected
- Interact with visible elements on a device
- Monitor interactions between app and system
- Simulate user interactions
- Requires instrumentation

What is instrumentation?

- A set of hooks in the Android system
- Loads test package and app into same process, allowing tests to call methods and examine fields
- Control components independently of app's lifecycle
- Control how Android loads apps

Benefits of instrumentation

- Tests can monitor all interaction with Android system
- Tests can invoke methods in the app
- Tests can modify and examine fields in the app independent of the app's lifecycle

Testing Environment & Setup

Install Android Support Library

1. In Android Studio choose T**ools > Android > SDK Manager**
2. Click **SDK Tools** and look for **Android Support Repository**
3. If necessary, update or install the library

Add dependencies to build.gradle

```
androidTestCompile 'com.android.support:support-annotations:24.1.1'
androidTestCompile 'com.android.support.test:runner:0.5'
androidTestCompile 'com.android.support.test:rules:0.5'
androidTestCompile 'org.hamcrest:hamcrest-library:1.3'
androidTestCompile
          'com.android.support.test.espresso:espresso-core:2.2.2'
androidTestCompile
          'com.android.support.test.uiautomator:uiautomator-v18:2.1.2'
```

Add defaultConfig dependencies

```
testInstrumentationRunner
    "android.support.test.runner.AndroidJUnitRunner"
```

Prepare your device

1. Turn on USB Debugging
2. Turn off all animations in **Developer Options > Drawing**
    - Window animation scale
    - Transition animation scale
    - Animator duration scale

Create tests

- Store in *module-name*`/src/androidTests/java/`
  - In Android Studio: app > java > *module-name* (androidTest)
- Create tests as JUnit classes

Writing Espresso Tests

Test class definition

```
@RunWith(AndroidJUnit4.class) // Required annotation for tests
@LargeTest  // Based on resources the test uses and time to run
public class ChangeTextBehaviorTest {}
```

@SmallTest—Runs in < 60s and uses no external resources

@MediumTest–Runs in < 300s, only local network

@LargeTest–Runs for a long time and uses many resources

@Rule specifies the context of testing

```
@Rule
public ActivityTestRule<MainActivity> mActivityRule =
    new ActivityTestRule<>(MainActivity.class);
```

@ActivityTestRule–Testing support for a single specified activity

@ServiceTestRule–Testing support for starting, binding, shutting down a service

@Before and @After set up and tear down

```
@Before
public void initValidString() {
    mStringToBetyped = "Espresso";
}
```

@Before–Setup, initializations

@After–Teardown, freeing resources

@Test method structure

```
@Test
public void changeText_sameActivity() {
    // 1. Find a View
    // 2. Perform an action
    // 3. Verify action was taken, assert result
}
```

"Hamcrest" simplifies tests

- “Hamcrest” an anagram of “Matchers”
- Framework for creating custom matchers and assertions
- Match rules defined declaratively
- Enables precise testing
- The Hamcrest Tutorial

Hamcrest Matchers

- ViewMatcher—find Views by id, content, focus, hierarchy
- ViewAction—perform an action on a view
- ViewAssertion—assert state and verify the result

Basic example test

```
@Test
public void changeText_sameActivity() {
    // 1. Find view by Id
    onView(withId(R.id.editTextUserInput))
    // 2. Perform action—type string and click button
    .perform(typeText(mStringToBetyped), closeSoftKeyboard());
    onView(withId(R.id.changeTextBt)).perform(click());
    // 3. Check that the text was changed
    onView(withId(R.id.textToBeChanged))
       .check(matches(withText(mStringToBetyped)));
    }
```

Finding views with onView

- `withId()`—find a view with the specified Android id
  - `onView(withId(R.id.editTextUserInput))`
- `withText()`—find a view with specific text
- `allOf()`—find a view to that matches multiple conditions. Find a visible list item with the given text:

```
onView(allOf(withId(R.id.word),
             withText("Clicked! Word 15"),
             isDisplayed()))
```

onView returns ViewInteraction object

- If you need to reuse the view returned by onView
- Make code more readable or explicit
- `check()` and `perform()` methods

```
ViewInteraction textView = onView(
    allOf(withId(R.id.word), withText("Clicked! Word 15"),
    isDisplayed()));
textView.check(matches(withText("Clicked! Word 15")));
```

Perform actions

- Perform an action on the view found by a ViewMatcher
- Can be any action you can perform on the view

```
// 1. Find view by Id
onView(withId(R.id.editTextUserInput))

// 2. Perform action—type string and click button
.perform(typeText(mStringToBetyped), closeSoftKeyboard());
onView(withId(R.id.changeTextBt)).perform(click());
```

Check result

- Asserts or checks the state of the view

```
// 3. Check that the text was changed
onView(withId(R.id.textToBeChanged))
    .check(matches(withText(mStringToBetyped)));
```

When a test fails

Test

```
onView(withId(R.id.text_message))
    .check(matches(withText("This is a failing test.")));
```

Result snippet

```
android.support.test.espresso.base.DefaultFailureHandler$Assertion
FailedWithCauseError: 'with text: is "This is a failing test."'
doesn't match the selected view.
Expected: with text: is "This is a failing test."
Got: "AppCompatTextView{id=2131427417, res-name=text_message ...
```

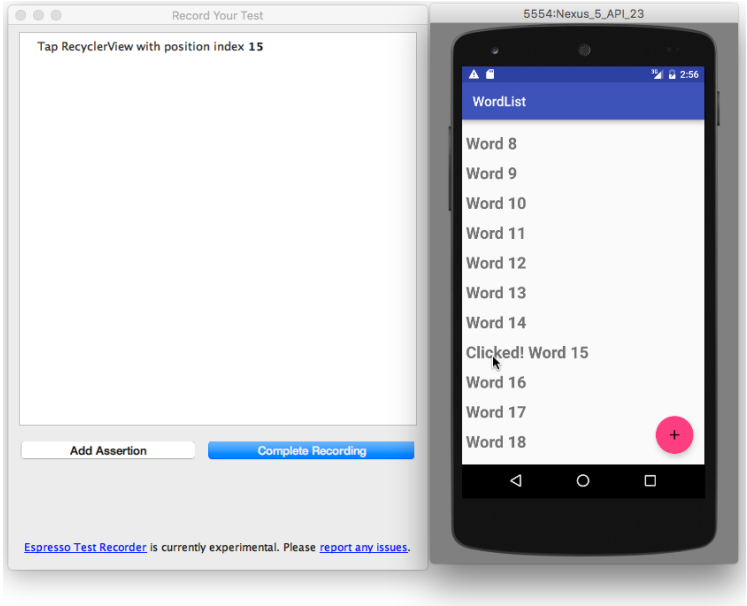
Recording Tests

Recording an Espresso test

- Android Studio 2.2
- Use app normally, clicking through the UI
- Editable test code generated automatically
- Add assertions to check if a view holds a certain value
- Record multiple interactions in one session, or record multiple sessions

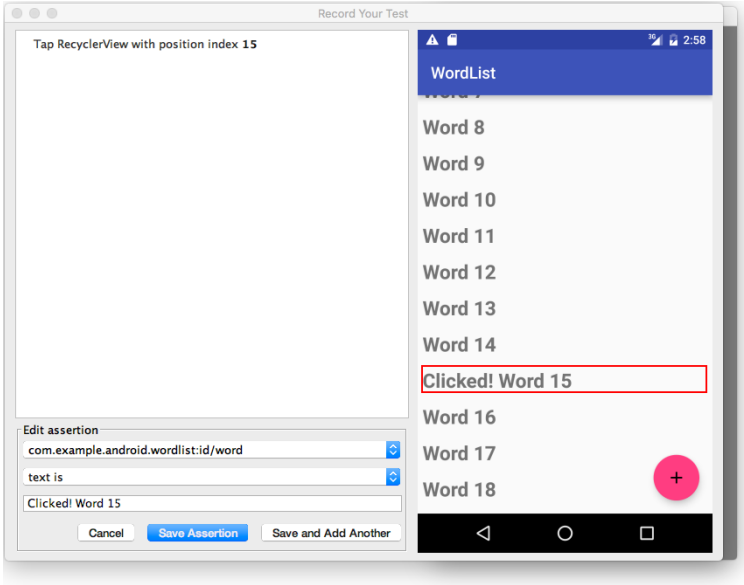
Start recording an Espresso test

1. **Run > Record Espresso Test**
2. Click **Restart app**, select target, and click **OK**
3. Interact with the app to do what you want to test



Add assertion to Espresso test recording

4. Click **Add Assertion** and select a UI element
5. Choose **text is** and enter the text you expect to see
6. Click **Save Assertion** and click **Complete Recording**



Learn more from developer docs

Android Studio Documentation

- Test Your App
- Espresso basics
- Espresso cheat sheet

Android Developer Documentation

- Best Practices for Testing
- Getting Started with Testing
- Testing UI for a Single App
- Building Instrumented Unit Tests
- Espresso Advanced Samples
- The Hamcrest Tutorial
- Hamcrest API and Utility Classes
- Test Support APIs

Learn even more

Android Testing Support Library

- Espresso documentation
- Espresso Samples

Videos

- Android Testing Support - Android Testing Patterns #1 (introduction)
- Android Testing Support - Android Testing Patterns #2 (onView view matching)
- Android Testing Support - Android Testing Patterns #3 (onData & adapter views)

Learn even more

- Google Testing Blog: Android UI Automated Testing
- Atomic Object: "Espresso – Testing RecyclerViews at Specific Positions"
- Stack Overflow: "How to assert inside a RecyclerView in Espresso?"
- GitHub: Android Testing Samples
- Google Codelabs: Android Testing Codelab

What's Next?

- Concept Chapter: 6.1 C Testing the User Interface
- Practical: 6.1 P Use Espresso to Test Your UI

END