5COSC005W MOBILE APPLICATION DEVELOPMENT
Lecture 2: Text and Scrolling Views – Buttons and Input Controls

Dr Dimitris C. Dracopoulos

Module Web page:

http://users.wmin.ac.uk/∼dracopd/DOCUM/courses/5cosc005w/5cosc005w.html

Text and scrolling views

TextView

TextView for text

- `TextView` is `View` subclass for single and multi-line text
- `EditText` is `TextView` subclass with editable text
- Controlled with layout attributes
- Set text:
  - Statically from string resource in XML
  - Dynamically from Java code

Formatting text in string resource

- Use `<b>` and `<i>` HTML tags for bold and italics
- All other HTML tags are ignored
- String resources: one unbroken line = one paragraph
- `\n` starts a new a line or paragraph
- Escape apostrophes and quotes with backslash (`\"`, `\'`)
- Escape any non-ASCII characters with backslash (`\`)

Creating TextView in XML

```
<TextView android:id="@+id/textview"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/my_story"/>
```

Common TextView attributes

`android:text`—text to display

`android:textColor`—color of text

`android:textAppearance`—predefined style or theme

`android:textSize`—text size in `sp`

`android:textStyle`—`normal`, `bold`, `italic`, or `bold|italic`

`android:typeface`—`normal`, `sans`, `serif`, or `monospace`

`android:lineSpacingExtra`—extra space between lines in `sp`

Formatting active web links

```
<string name="article_text">... www.rockument.com ...</string>

<TextView
    android:id="@+id/article"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:autoLink="web"
    android:text="@string/article_text"/>
```

autoLink values:"web", "email", "phone", "map", "all"

Don't use HTML for a web link in free-form text

Creating TextView in Java code

```
TextView myTextview = new TextView(this);
myTextView.setWidth(LayoutParams.MATCH_PARENT);
myTextView.setHeight(LayoutParams.WRAP_CONTENT);
myTextView.setMinLines(3);
myTextView.setText(R.string.my_story);
myTextView.append(userComment);
```

ScrollView

What about large amounts of text?

- News stories, articles, etc...
- To scroll a `TextView`, embed it in a `ScrollView`
- Only *one* `View` element (usually `TextView`) allowed in a `ScrollView`
- To scroll multiple elements, use one `ViewGroup` (such as `LinearLayout`) within the `ScrollView`

ScrollView for scrolling content

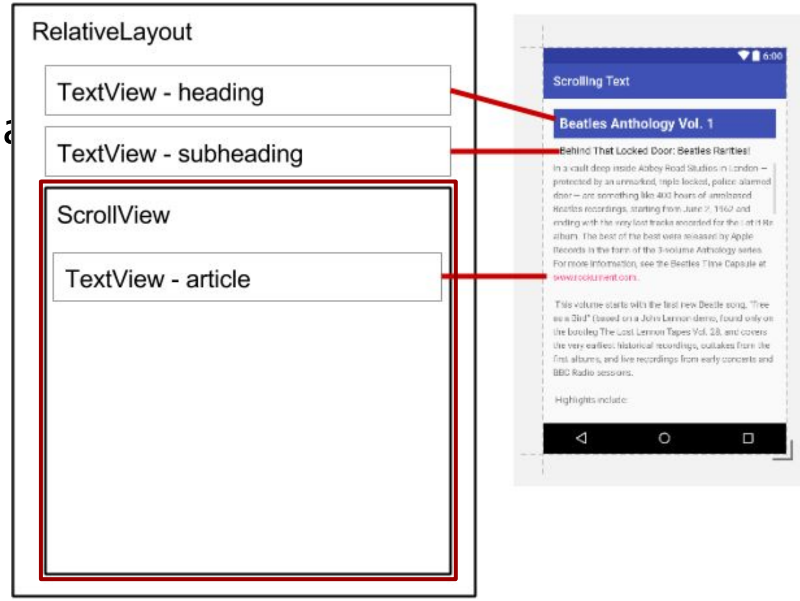
- `ScrollView` is a subclass of `FrameLayout`
- Holds all content in memory
- Not good for long texts, complex layouts
- Do not nest multiple scrolling views
- Use `HorizontalScrollView` for horizontal scrolling
- Use a `RecyclerView` for lists

ScrollView layout with one TextView

```
<ScrollView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_below="@id/article_subhea

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    .../>

</ScrollView>
```



Beatles Anthology Vol. 1

Behind That Locked Door: Beatles Rarities!

In a vault deep inside Abbey Road Studios in London — protected by an unmarked, triple locked, police alarmed door — are something like 400 hours of unreleased Beatles recordings, starting from June 2, 1962 and ending with the very last tracks recorded for the Let It Be album. The best of the best were released by Apple Records in the form of the 3-volume Anthology series. For more information, see the Beatles Time Capsule at www.rockument.com.

This volume starts with the first new Beatle song, "Free as a Bird" (based on a John Lennon demo, found only on the bootleg The Lost Lennon Tapes Vol. 28, and covers the very earliest historical recordings, outtakes from the first albums, and live recordings from early concerts and BBC Radio sessions.

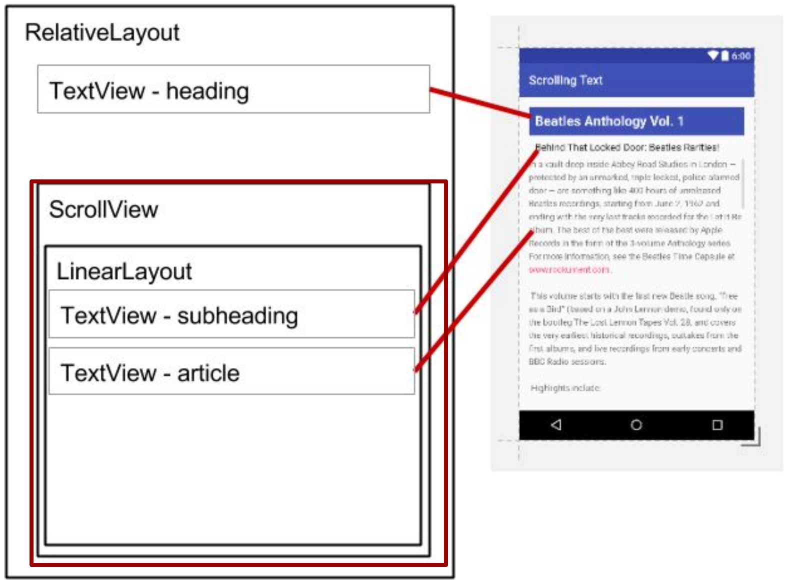
Highlights include:

ScrollView layout with a view group

```
<ScrollView ...
   <LinearLayout
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:orientation="vertical">

       <TextView
           android:id="@+id/article_subheading"
           .../>

       <TextView
           android:id="@+id/article" ... />
   </LinearLayout>
</ScrollView>
```



ScrollView with image and button

```
<ScrollView...>
    <LinearLayout...>
        <ImageView.../>
        <Button.../>
        <TextView.../>
    </LinearLayout>
</ScrollView>
```

One child of ScrollView which can be a layout

Children of the layout

Buttons and clickable images

User interaction

Users expect to interact with apps

- Tapping or clicking, typing, using gestures, and talking
- Buttons perform actions
- Other UI elements enable data input and navigation

User interaction design

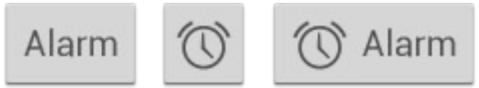
Important to be obvious, easy, and consistent:

- 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

Buttons

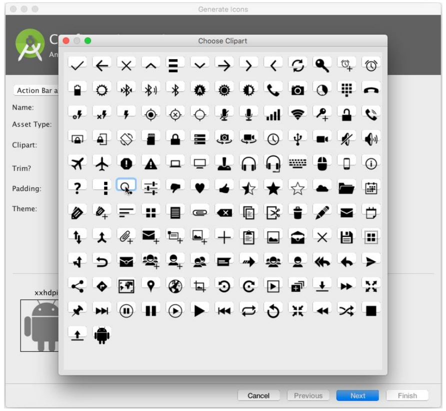
Button

- View that responds to tapping (clicking) or pressing
- Usually text or visuals indicate what will happen when tapped
- State: normal, focused, disabled, pressed, on/off



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**

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" />
```

android:onClick

Setting listener with onClick callback

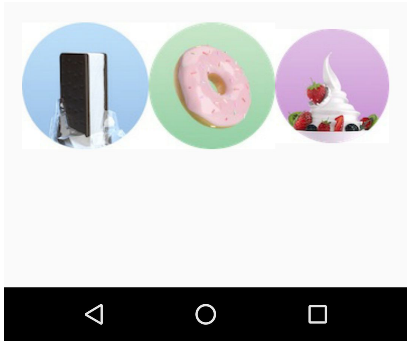
```
Button button = findViewById(R.id.button);

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

Clickable images

ImageView

- `ImageView` with `android:onClick` attribute
- Image for `ImageView` in **app>src>main>res>drawable** folder in project



Responding to ImageView taps

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

```
<ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/donut_circle"
        android:onClick="orderDonut"/>
```

android:onClick

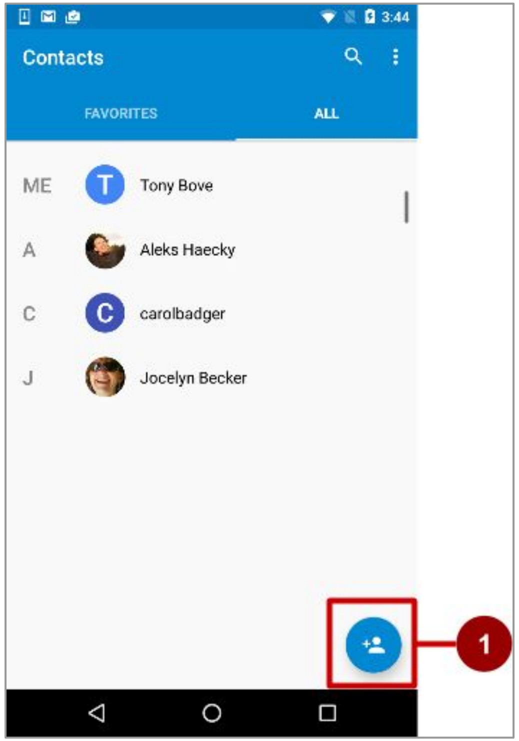
Floating action button

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



Using FABs

- Start with Basic Activity template
- Layout:

```
<com.google.android.material.floatingactionbutton.FloatingActionButton
        android:id="@+id/fab"
        android:layout_gravity="bottom|end"
        android:layout_margin="@dimen/fab_margin"
        android:src="@drawable/ic_fab_chat_button_white"

        .../>
```

FAB size

- 56 x 56 dp by default
- Set mini size (30 x 40 dp) with `app:fabSize` attribute:
    - `app:fabSize="mini"`
- Set to 56 x 56 dp (default):
    - `app:fabSize="normal"`

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!

Detect gestures

Classes and methods are available to help you handle gestures.

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

Input Controls

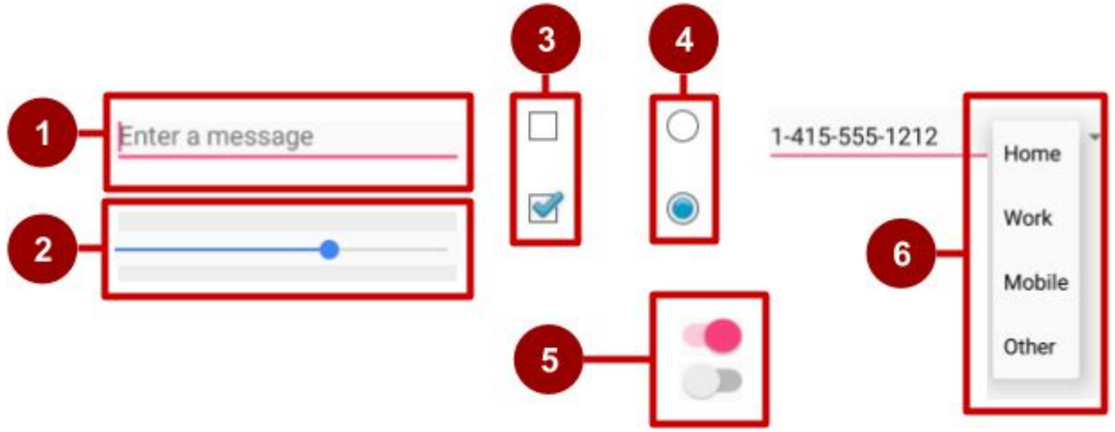
Overview of input Controls

Accepting user input

- Freeform text and numbers: `EditText` (using keyboard)
- Providing choices: `CheckBox`, `RadioButton`, `Spinner`
- Switching on/off: `Toggle`, `Switch`
- Choosing value in range of values: `SeekBar`

Examples of input controls

1. EditText
2. SeekBar
3. CheckBox
4. RadioButton
5. Switch
6. Spinner



How input controls work

1. Use `EditText` for entering text using keyboard
2. Use `SeekBar` for sliding left or right to a setting
3. Combine `CheckBox` elements for choosing more than one option
4. Combine `RadioButton` elements into `RadioGroup` — user makes only one choice
5. Use `Switch` for tapping on or off
6. Use `Spinner` for choosing a single item from a list

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`

View 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

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

Set focus explicitly

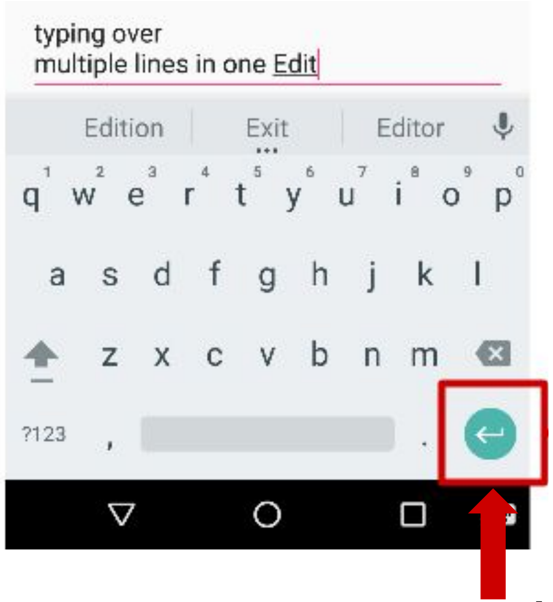
Use methods of the View class to set focus

- 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

Freeform text and numbers

EditText for multiple lines of text

- EditText default
- Alphanumeric keyboard
- Suggestions appear
- Tapping **Return** (**Enter**) key starts new line

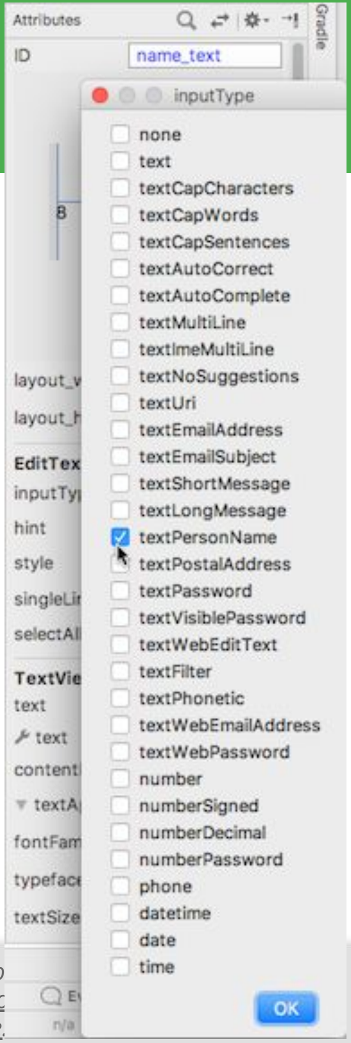


Return key

Customize with inputType

- Set in Attributes pane of layout editor
- XML code for EditText:

```
<EditText
    android:id="@+id/name_field"
    android:inputType =
                 "textPersonName"
    ...
```



Attributes

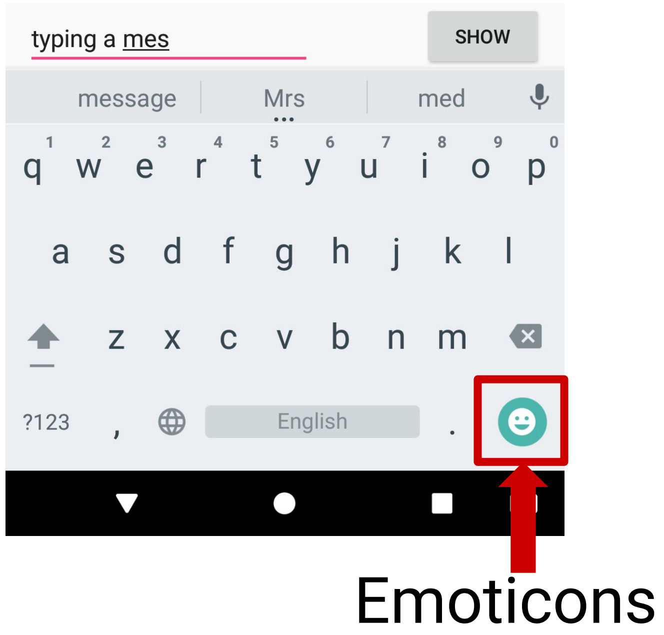
ID name_text

inputType

- none
- text
- textCapCharacters
- textCapWords
- textCapSentences
- textAutoCorrect
- textAutoComplete
- textMultiLine
- textImeMultiLine
- textNoSuggestions
- textUri
- textEmailAddress
- textEmailSubject
- textShortMessage
- textLongMessage
- [✓ textPersonName]
- textPostalAddress
- textPassword
- textVisiblePassword
- textWebEditText
- textFilter
- textPhonetic
- textWebEmailAddress
- textWebPassword
- number
- numberSigned
- numberDecimal
- numberPassword
- phone
- datetime
- date
- time

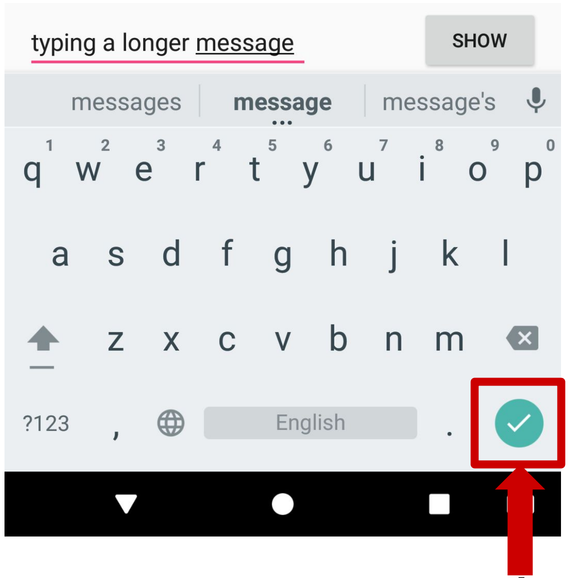
EditText for message

- `android:inputType`
  `="textShortMessage"`
- Single line of text
- Tapping Emoticons key changes keyboard to emoticons



EditText for single line

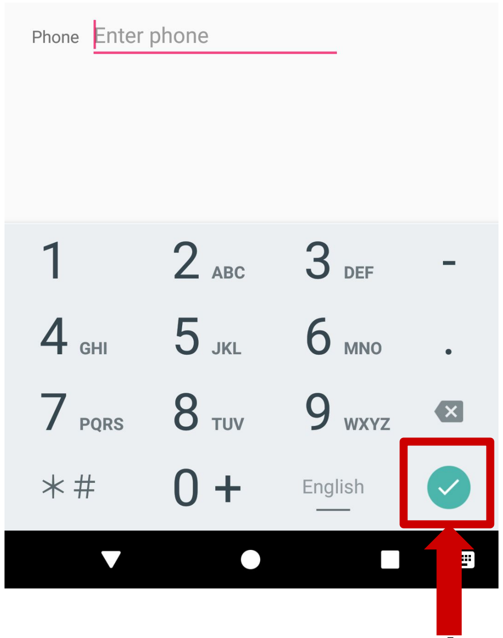
- Both work:
  - `android:inputType ="textLongMessage"`
  - `android:inputType ="textPersonName"`
- Single line of text
- Tapping **Done** key advances focus to next View



Done key

EditText for phone number entry

- `android:inputType ="phone"`
- Numeric keypad (numbers only)
- Tapping **Done** key advances focus to next View



Done key

Getting text

- Get the EditText object for the EditText view

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

- Retrieve the CharSequence and convert it to a string

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

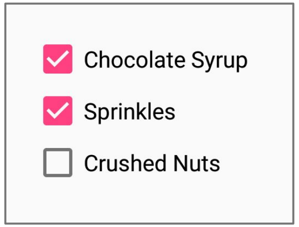
Common input types

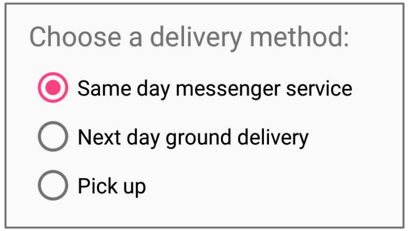
- `textCapCharacters`: Set to all capital letters
- `textCapSentences`: Start each sentence with a capital letter
- `textPassword`: Conceal an entered password
- `number`: Restrict text entry to numbers
- `textEmailAddress`: Show keyboard with @ conveniently located
- `phone`: Show a numeric phone keypad
- `datetime`: Show a numeric keypad with a slash and colon for entering the date and time

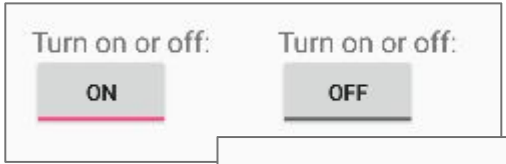
Providing choices

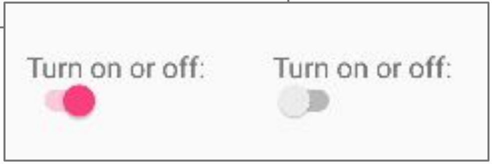
UI elements for providing choices

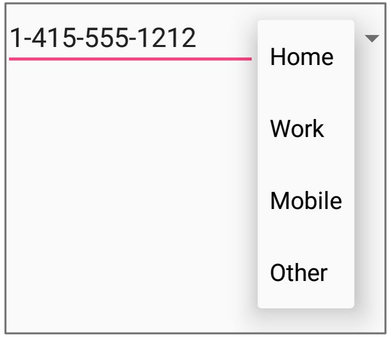
- CheckBox and RadioButton
- ToggleButton and Switch
- Spinner





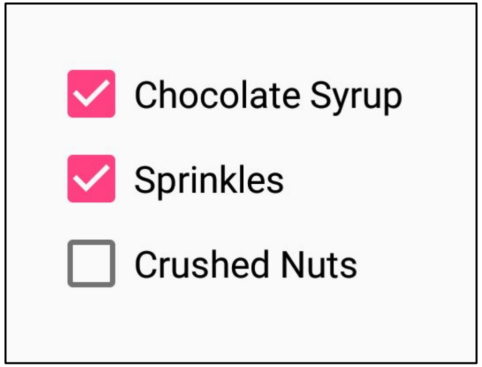






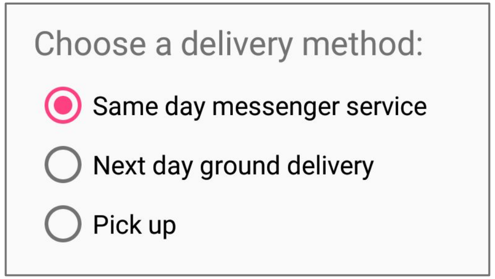
CheckBox

- 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



RadioButton

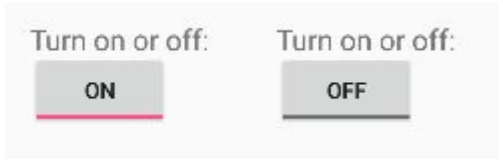
- Put RadioButton elements in a RadioGroup in a vertical list (horizontally if labels are short)
- User can select only one of the choices
- Checking one unchecks all others in group
- Each RadioButton can have onClick handler
- Commonly used with a **Submit** button for the RadioGroup


Choose a delivery method:

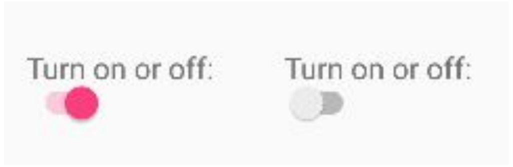
- [✓ Same day messenger service]
- Next day ground delivery
- Pick up

Toggle buttons and switches

- User can switch between on and off
- Use `android:onClick` for click handler



Toggle buttons



Switches

Alternative resources

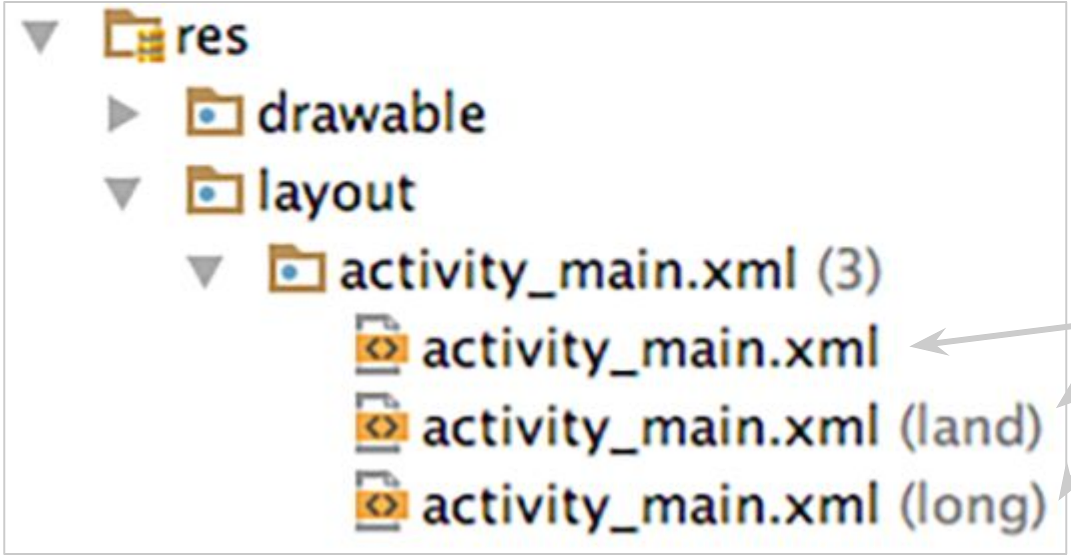
What are alternative resources?

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

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-sw**600**dp/dimens.xml
    - Does not change with orientation
- Android uses resource closest to (without exceeding) the device's smallest width

Smallest Width Qualifier Examples

- **320dp**: a typical phone screen (240x320 ldpi, 320x480 mdpi, 480x800 hdpi, etc)
- **480dp**: a large phone screen ~5" (480x800 mdpi)
- **600dp**: a 7" tablet (600x1024 mdpi)
- **720dp**: a 10" tablet (720x1280 mdpi, 800x1280 mdpi, etc)

Dp vs DPI

- **DPI**: Dots (pixels) per inch
- **DP**: Density independent pixels

$$dp = \frac{pixels * 160}{dpi} \tag{1}$$

ldpi	120dpi
mdpi	160dpi
hdpi	240dpi
xhdpi	320dpi
xxhdpi	480dpi
xxxhdpi	640dpi