Android Developer Fundamentals V2

User Interaction

Lesson 4

4.2 Input Controls

Contents

- Overview of input controls
- View focus
- Freeform text and numbers
- Providing choices

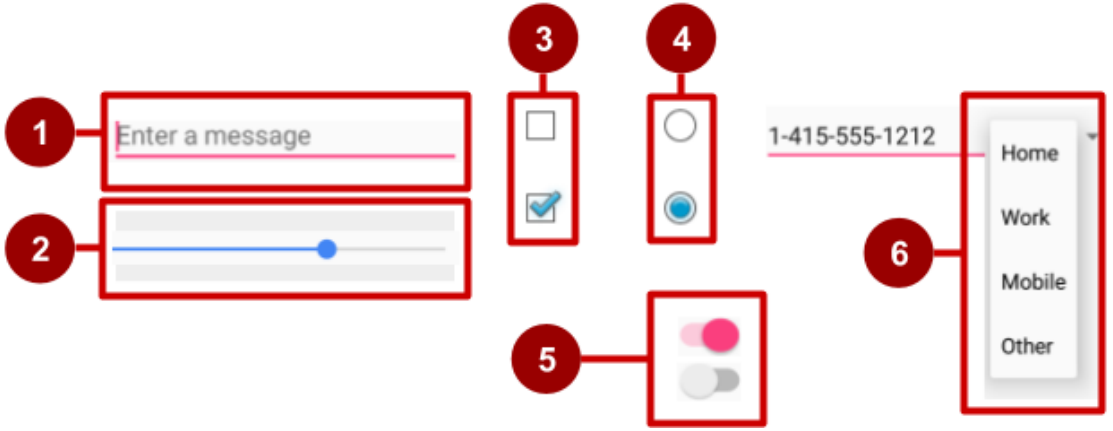
Overview of input Controls

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

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

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!

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"
```

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

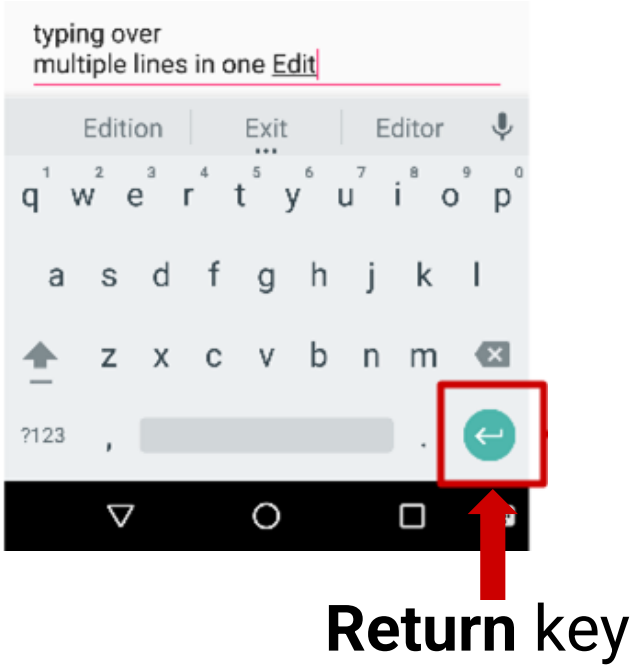
Find the view with focus

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

Freeform text and numbers

EditText for multiple lines of text

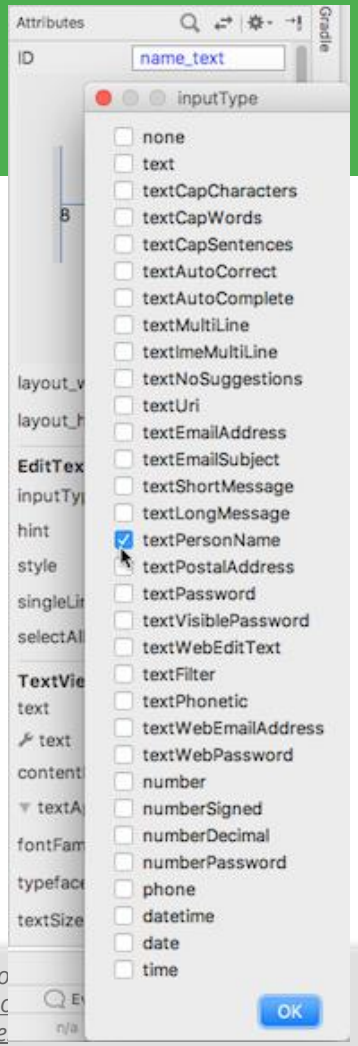
- EditText default
- Alphanumeric keyboard
- Suggestions appear
- Tapping **Return** (**Enter**) key starts new line



Customize with inputType

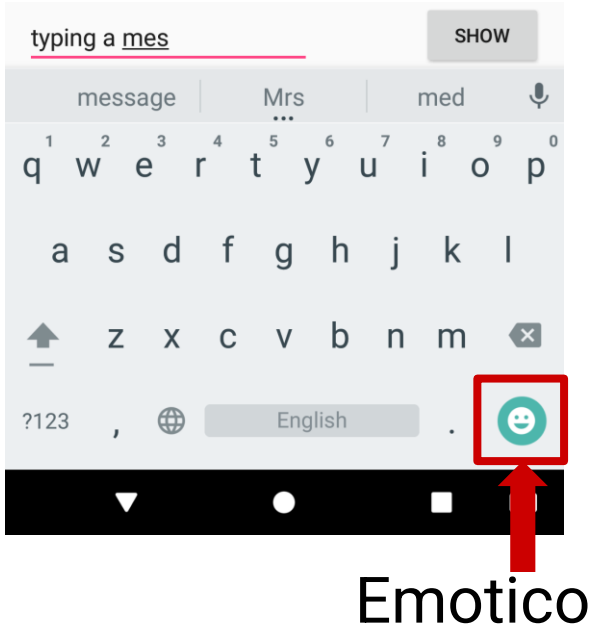
- Set in Attributes pane of layout editor
- XML code for EditText:

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



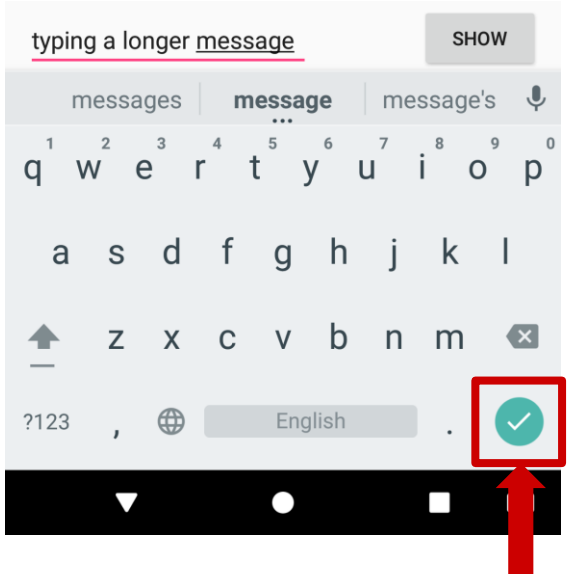
EditText for message

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



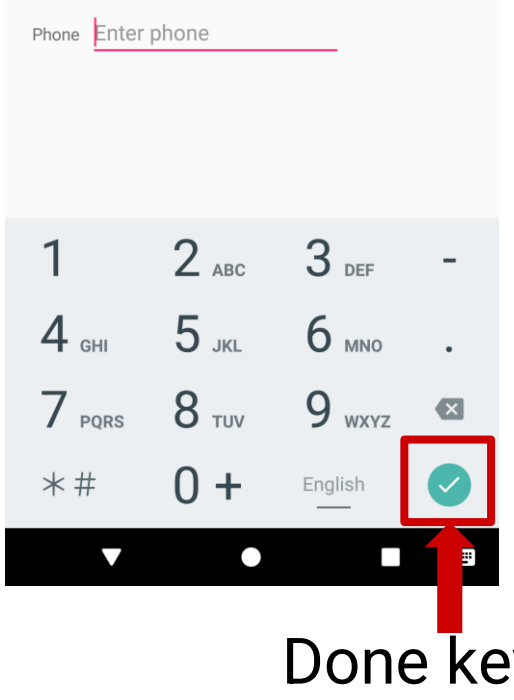
EditText for single line

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



EditText for phone number entry

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



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();
```

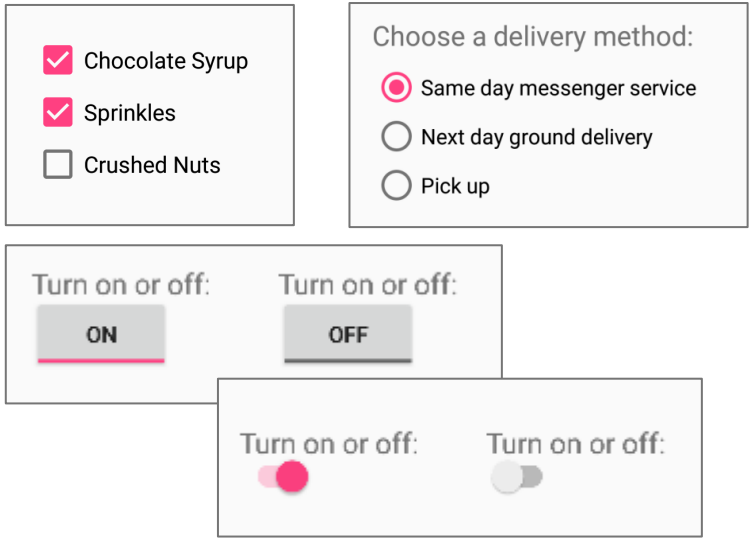
Common input types

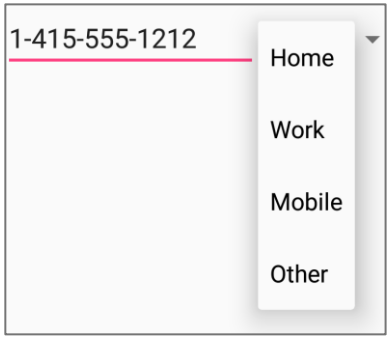
- `textCapCharacters`: Set to all capital letters
- `textCapSentences`: Start each sentence with a capital letter
- `textPassword`: Conceal an entered password
- `number`: Restrict text entry to numbers
- `textEmailAddress`: Show keyboard with @ conveniently located
- `phone`: Show a numeric phone keypad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

Providing choices

UI elements for providing choices

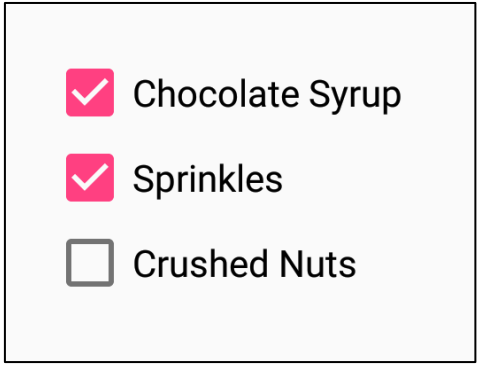
- CheckBox and RadioButton
- ToggleButton and Switch
- Spinner





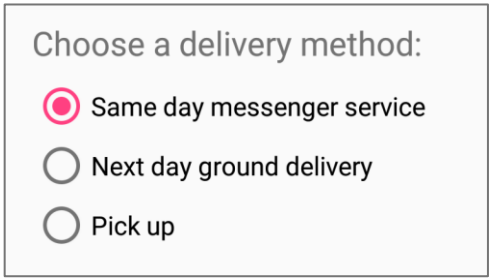
CheckBox

- 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



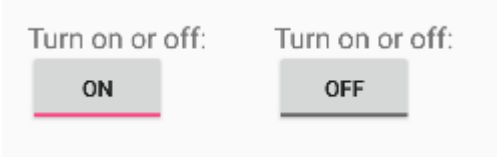
RadioButton

- 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`

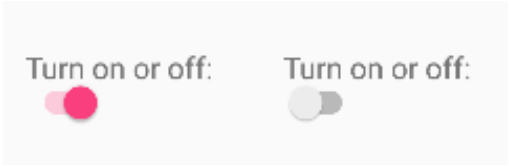


Toggle buttons and switches

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



Toggle buttons



Switches

Learn more

- Input Controls
- Radio Buttons
- Specifying the Input Method Type
- Handling Keyboard Input
- Text Fields
- Spinners

What's Next?

- Concept Chapter: 4.2 Input controls
- Practical: 4.2 Input controls

END