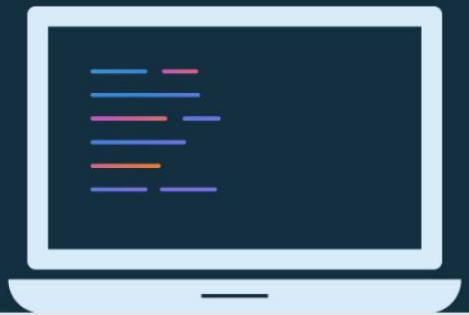
Lezione 2.3 Componenti di layout



No surprises - Radiohead

In questa lezione

- TextView
- ScrollView
- Buttons & clickable images
- Input controls

TextView

TextView per il testo

- `TextView` è una sottoclasse di `View` per rappresentare testo (come singola linea o multi-linea)
- `EditText` è una `TextView` con testo editabile
- Sono controllabili con attributi di layout
- Il testo può essere settato:
  - Staticamente da una string resource in XML
  - dinamicamente dal codice

Formattazione del testo nelle string resource

- Usa i tag HTML `<b>` e `<i>` per **grassetto** e *corsivo*
- Tutti gli altri tag HTML vengono ignorati
- String resources: one unbroken line = one paragraph
- `\n` inizia una nuova linea
- L’escape per apostrofi, doppie virgolette e caratteri non-ASCII avviene tramite backslash (`\"`, `\'`)

Creare TextView in XML

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

Attributi comuni per TextView

Android:**text**—testo da visualizzare

android:**textColor**—colore del testo

android:**textAppearance**—stile predefinito o tema

android:**textSize**—dimensione del testo in sp (simile al dp ma per il testo)

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

android:**typeface**—normal, sans, serif, o monospace

android:**lineSpacingExtra**—spazio extra tra linee in sp

Formattazione di web link attivi

```
<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"/>
```

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

Nota: non si usa direttamente HTML per un link web

Editare una TextView

```
myTextView.minLines = 3
myTextView.text=getString(R.string.my_story)
myTextView.append(userComment)
myTextView.layoutParams =
    LinearLayout.LayoutParams(ViewGroup.LayoutParams.WRAP_CO
NTENT, ViewGroup.LayoutParams.WRAP_CONTENT)
```

ScrollView

Che fare se abbiamo troppo testo?

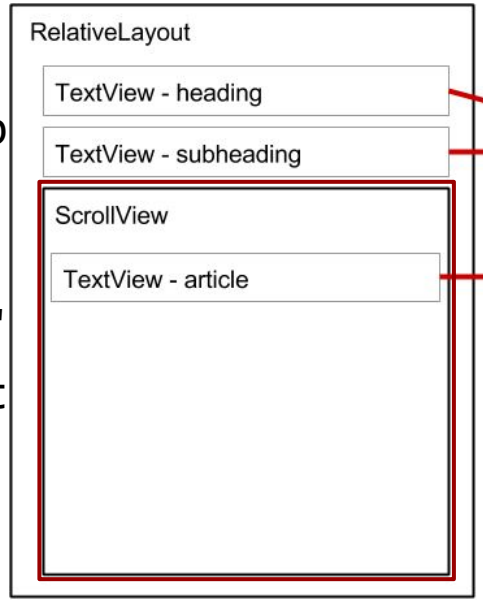
- Per scorrere una `TextView` è sufficiente incorporarla in una `ScrollView`
- Una `ScrollView` può contenere solo *una* `View` (tipicamente una `TextView`)
- Per scorrere elementi multipli, usa una `ViewGroup` (ad esempio un `LinearLayout`) all'interno di una `ScrollView`

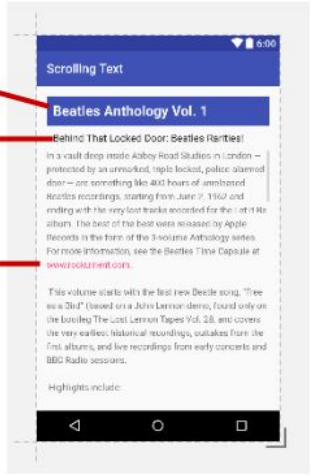
ScrollView

- `ScrollView` è sottoclasse di `FrameLayout`
- Gestisce tutto il contenuto in memoria, pertanto non è efficiente per testo eccessivamente lungo o layout complessi
- Non si possono annidare ScrollView multiple
- Usa `HorizontalScrollView` per scorrere orizzontalmente
- Usa una `RecyclerView` per le liste

Il layout di ScrollView con una TextView

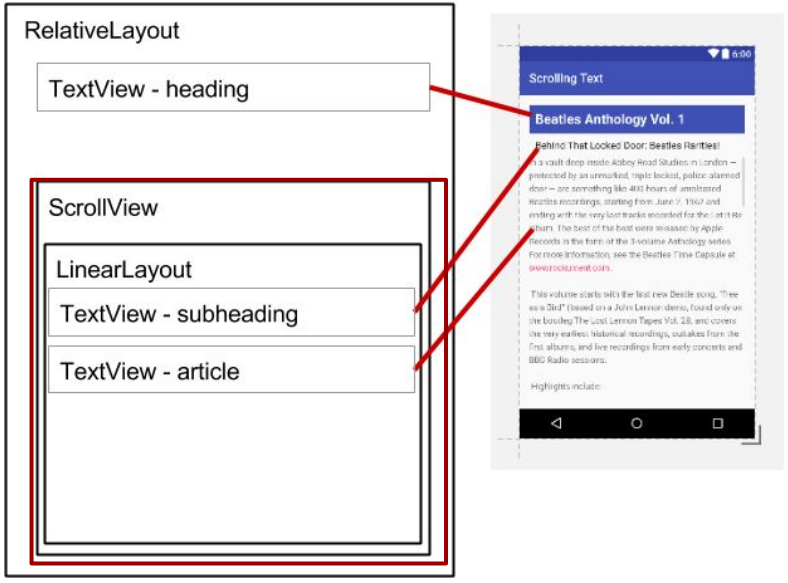
```
<ScrollView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_below="@id/article_sub

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

</ScrollView>
```





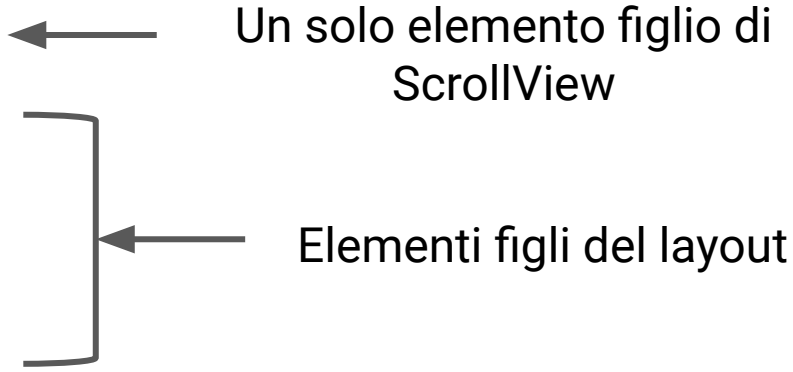
Il layout di ScrollView con una ViewGroup

```
<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 con immagine e bottone

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



Progettare la User Interaction

È importante che sia ovvia, semplice e consistente:

- Pensiamo a come gli utenti useranno l’app
- Minimizzare i passi per eseguire delle azioni
- Usare elementi dell’UI che sono facilmente accessibili, comprensibili ed utilizzabili
- Seguire le best practice di Android

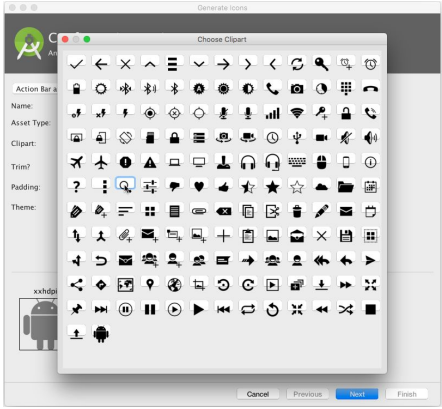
Button

Button

- Una View che risponde al tapping (clicking) o alla pressione
- Tipicamente corredati di testo o immagini che indicano cosa accadrà se viene premuto
- Può assumere diversi stati: normal, focused, disabled, pressed, on/off

Immagini per i Button

1. Clicca col destro su app/res/drawable
2. Scegli **New > Image Asset**
3. Scegli **Action Bar and Tab Items** dal drop down menu
4. Clicca **Clipart:** image (il logo Android)



Alternativa:

2. Scegli **New > Vector Asset**

Rispondere al click su bottone

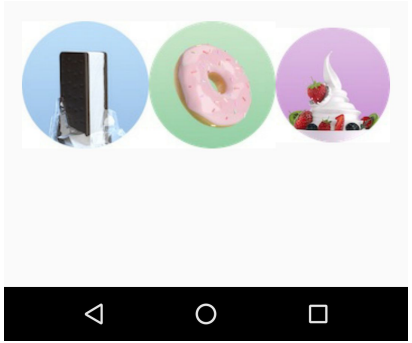
- *Nel codice*: usa l'event listener `OnClickListener`
- *Nel XML*: usa l'attributo `android:onClick`

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

Clickable image

ImageView

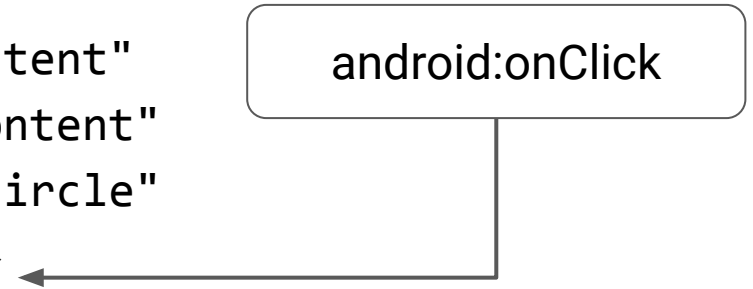
- Una `ImageView` con un attributo `android:onClick`
- L'immagine per l'`ImageView` può essere selezionata dalla directory **app>src>main>res>drawable**



Rispondere al tap su una ImageView

- *Nel codice*: usa un event listener `OnClickListener`
- *Nel XML*: usa un attributo `android:onClick`

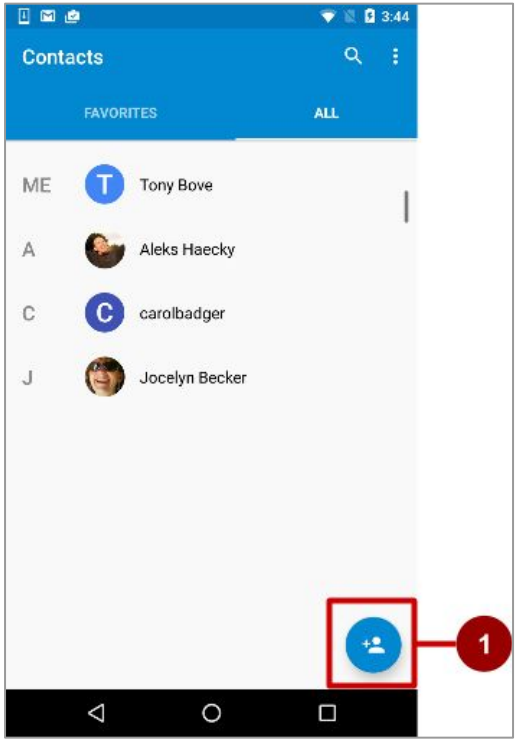
```
<ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/donut_circle"
        android:onClick="orderDonut"/>
```



Floating Action Buttons (FAB)

- Bottone circolare posizionato in cima al layout
- Utilizzati per segnalare un'azione particolarmente importante per una certa schermata
- Uno per schermata

Ad esempio: bottone **Aggiungi contatto** nella rubrica



Usare FABs

- 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"
        .../>
```

Dimensioni di un FAB

- 56 x 56 dp di default
- Setta una dimensione rimpicciolita (30 x 40 dp):
    - `app:fabSize="mini"`
- Set una dimensione 56 x 56 dp (default):
    - `app:fabSize="normal"`

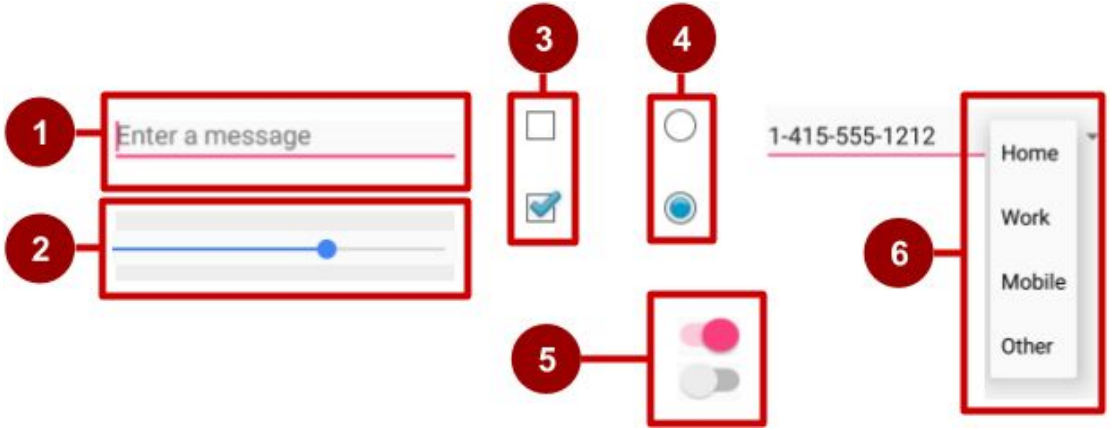
Input Controls

Tipi di user input

- Testo e numeri: `EditText` (usando la tastiera)
- Input di scelta: `CheckBox`, `RadioButton`, `Spinner`
- Switching on/off: `Toggle`, `Switch`
- Scegliere un valore in un range: `SeekBar`

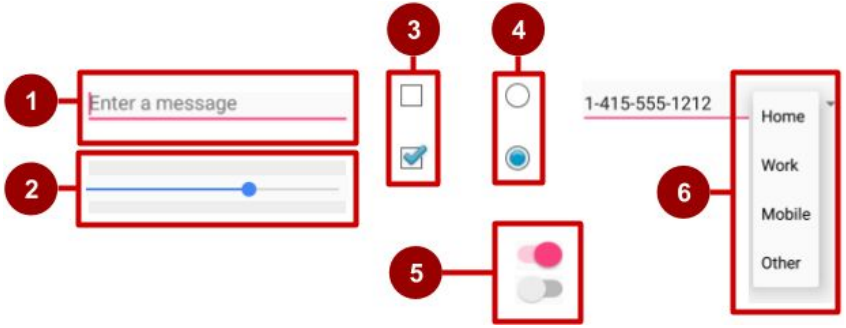
View per l'input

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



Come funzionano i controller di input

1. `EditText` per inserire testo da tastiera
2. `SeekBar` per settare un valore tramite sliding a destra o sinistra
3. `CheckBox` per scegliere tra diverse opzioni
4. Elementi `RadioButton` (raggruppati in `RadioGroup`) per scegliere una sola opzione
5. `Switch` per valori on/off
6. `Spinner` per scegliere un elemento da una lista



View come base per i controlli di input

- La classe View è il componente base per tutti gli elementi di UI, inclusi i controlli di input
  - fornisce elementi UI interattivi
  - Interazione di base tramite `android:onClick`

Focus

- Una View che riceve un input ha un "Focus"
- Solo una View alla volta può averlo
- Il focus disambigua da quale View proviene l’input
- Il focus può essere assegnato in vari modi:
  - L’utente clicca su una View
  - Oppure l’app può guidare l’utente da una view all’altra usando **Return**, **Tab**, o le frecce
  - Oppure chiamando `requestFocus()` su qualunque View che può ricevere un focus

Clickable vs focusable

Clickable—Una View può rispondere al click o al tap

Focusable—Una View può ricevere un focus per accettare dell'input

I controlli di input come una tastiera inviano l'input alla view che ha il focus in quel momento

Qual’è la prossima View che avrà il focus?

- Appena l’utente clicca in un punto, la view più in primo piano riceverà il focus
- Appena l’utente invia l’input, il focus si muove alla più vicina view —la priorità è da sinistra a destra, da sopra a sotto
- Il focus può cambiare anche se l’utente interagisce con un controllo direzionale

Guidare l’utente

- E’ bene indicare visivamente quale view ha il focus per far sapere agli utenti dove verrà inviato l’input
- Ciò aiuta anche a navigare tra le view per inserire dati
- Cercare di seguire un criterio predicibile e logico - no surprises!

Guidare il focus

- Organizza i controlli di input nel layout da sinistra a destra e dall'alto in basso in base all'ordine di focus voluto
- O posiziona i controlli di input dentro un viewgroup
- O specifica l'ordine in XML

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

Setta il focus in modo esplicito

Usa proprietà/metodi della classe View per settare il focus:

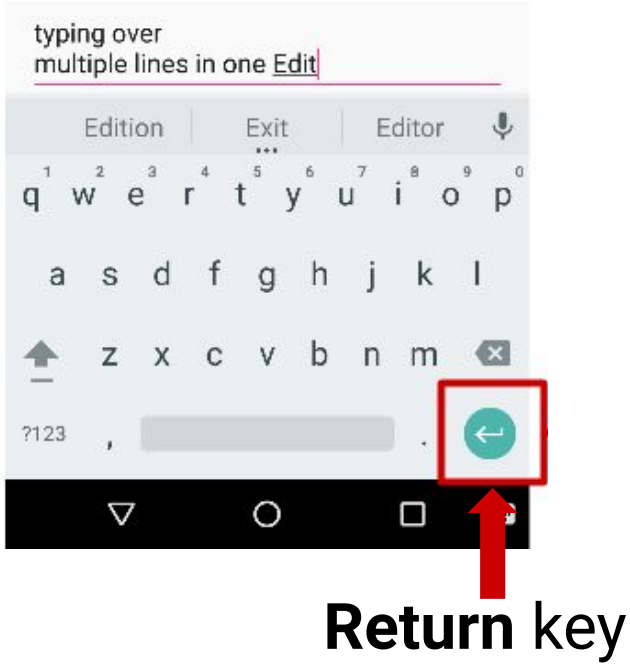
- focusable dichiara se la view può avere un focus
- requestFocus() dà il focus a una view
- setOnFocusChangeListener() setta un listener da usare quando la view acquisisce (o perde) il focus
- setOnFocusChangedListener() chiamata quando il focus cambia

Trovare quale view ha il focus

- Activity.currentFocus
- ViewGroup.focusedChild

EditText per linee di testo multiple

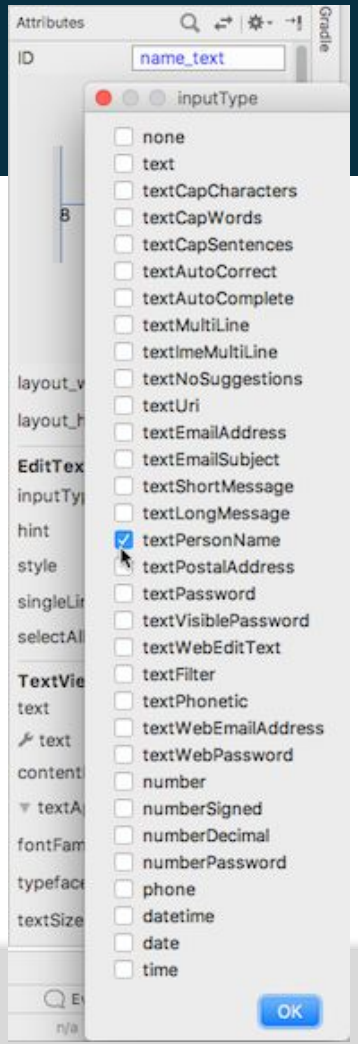
- È l'EditText di default
- Tastiera alfanumerica
- Appaiono suggerimenti
- Cliccando **Return** (**Enter**) inizia una nuova linea



Personalizza con l'inputType

- Nel pannello Attributes del layout editor
- Codice XML per EditText:

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

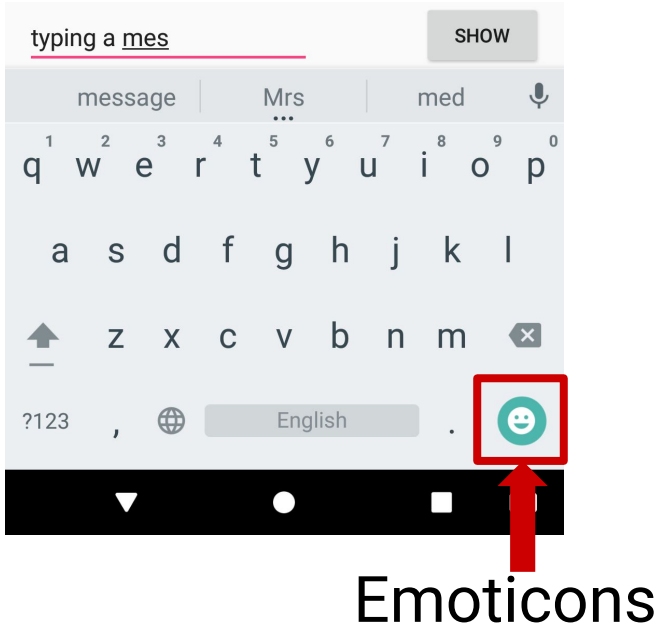


Tipi di input comuni

- `textCapCharacters`: tutto maiuscolo
- `textCapSentences`: prima lettera maiuscola
- `textPassword`: nasconde la password inserita
- `number`: restringe il testo solo ai numeri
- `textEmailAddress`: mostra la tastiera con il simbolo @ bene in vista
- `phone`: mostra una tastiera numerica per inserire un numero di telefono
- `datetime`: mostra una tastiera numerica con “/” e “:” per inserire data e ora

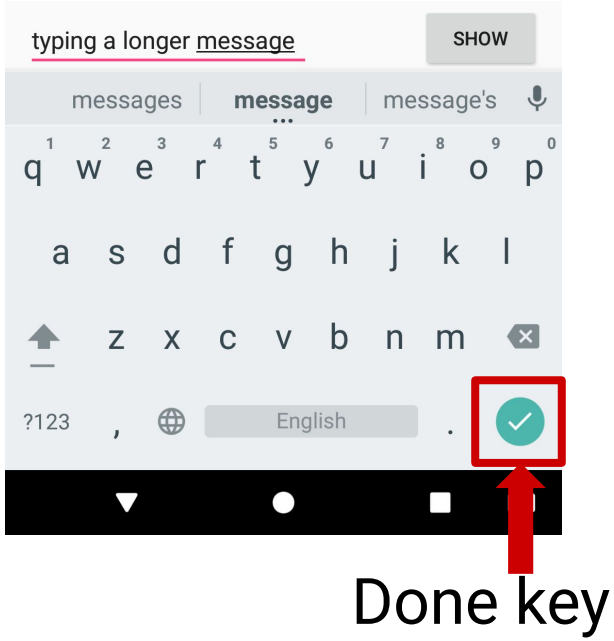
EditText per un messaggio

- `android:inputType ="textShortMessage"`
- Singola linea di testo
- Cliccando sul bottone Emoticons attiva la scheda corrispondente



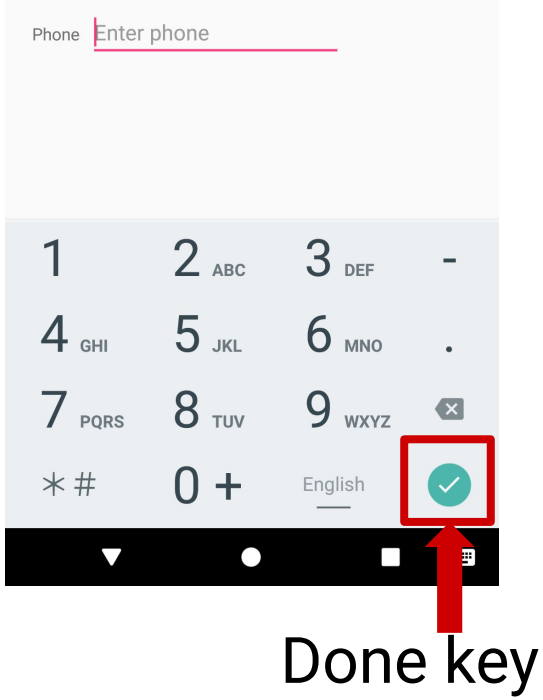
EditText per una singola linea

- Due opzioni:
  - `android:inputType="textLongMessage"`
  - `android:inputType="textPersonName"`
- Singola linea di testo
- Cliccando **Done** sposta il focus sulla prossima view



EditText per un numero di telefono

- `android:inputType ="phone"`
- Tastiera numerica
- Cliccando **Done** sposta il focus sulla prossima view



Leggere testo

- Ottenere un riferimento all'oggetto EditText:

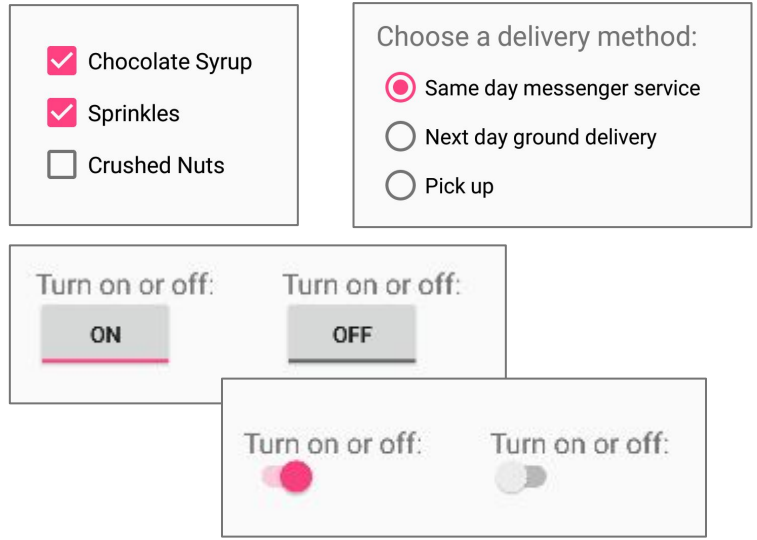
```
simpleEditText =
      findViewById<EditText>(R.id.edit_simple)
```

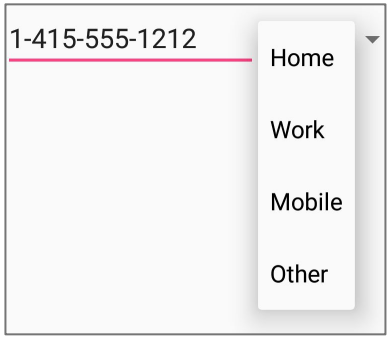
- Leggere il CharSequence contenuto e convertirlo in String

```
strValue: String =
       simpleEditText.text.toString()
```

Elementi UI per la scelta

- `SeekBar`
- `CheckBox` e `RadioButton`
- `ToggleButton` e `Switch`
- `Spinner`



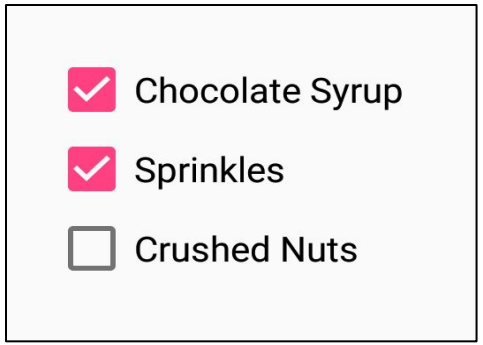


SeekBar

- Una barra utilizzata per effettuare regolazioni all'interno di un range
- Di default contiene valori (interi) da 0 a 100
- Event handling tramite i metodi `onProgressChanged` (quando viene cambiato il valore), `onStartTrackingTouch` e `onStopTrackingTouch` (quando si inizia/termina di muoverlo)
- Esempi: https://davidetech.blogspot.com/2020/03/android-studio-seekbar-1.html

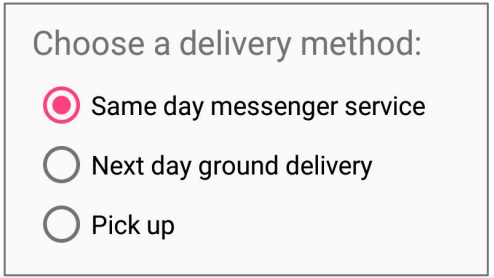
CheckBox

- L'utente può selezionare un numero qualsiasi di opzioni
- Selezionarne una non deseleziona le altre
- L'ordine delle opzioni è tipicamente verticale
- Usato comunemente con un bottone **Submit**
- Ogni `CheckBox` è una `View` e può avere
  Un handler `onClick`



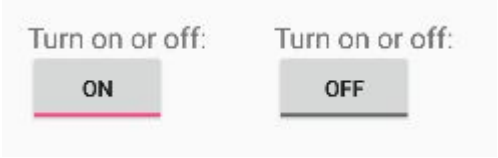
RadioButton

- Gli elementi `RadioButton` sono inseriti in un `RadioGroup` come lista verticale (orizzontale se le etichette sono corte)
- Una sola opzione selezionabile
- La selezione di una deseleziona le altre in un gruppo
- Usato comunemente con un bottone **Submit**
- Ciascun `RadioButton` può avere un handler `onClick`

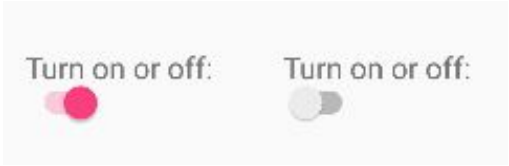


Bottoni toggle e switch

- L'utente può cambiare il valore tra on e off
- Utilizzano `android:onClick` per la gestione del click



Toggle buttons



Switches

Per approfondire

- Input Controls
- Radio Buttons
- Specifying the Input Method Type
- Handling Keyboard Input
- Text Fields
- Spinners
- Esempi e coding challenge: Input controls