Android Programming

Metodi di input

Michele Tosi

Anuar Elio Magliari

Marina Sotiropoulos

Eventi di input

- Un `evento di input` è una particolare azione dell'utente a cui è associata una risposta da parte dell'applicazione.
- Per gestire tali eventi, all'interno delle varie classi View esistono diversi `metodi di callback`.
- Questi metodi vengono chiamati dal `framework Android` quando si verifica la rispettiva azione su quell'oggetto.
- Il framework Android mantiene una `coda di eventi` in cui essi vengono posti man mano che si verificano e dalla quale vengono rimossi in modalità FIFO, ovvero in ordine di entrata.

Event listener

- Un `event listener` è un'interfaccia nella classe View che contiene un singolo metodo di callback.
- Di seguito sono riportati gli event listener presenti all'interno del framework Android e i relativi metodi callback ad essi associati:
  - `onClickListener` -> `onClick()`
  - `onLongClickListener` -> `onLongClick()`
  - `onFocusChangeListener` -> `onFocusChange()`
  - `onKeyListener` -> `onKey()`
  - `onTouchListener` -> `onTouch()`
  - `onCreateContextMenuListener` -> `onCreateContextMenu()`

Event listener - 2

- Tra i metodi callback dell'event listener ve ne sono alcuni che restituiscono un valore booleano.
- Valore legato al fatto di aver consumato o meno l'evento.
- I metodi in questione sono:
  - `onLongClick()`
  - `onKey()`
  - `onTouch()`
- Android chiamerà prima i gestori degli eventi e poi i gestori predefiniti appropriati dalla definizione della classe. Come tale, il ritorno `true` da questi event listener fermerà la propagazione dell'evento ad altri ascoltatori e bloccherà anche il callback al gestore di eventi predefinito nella View.
- Quindi siate certi di voler terminare o meno l'evento quando ritorna true.

Esempio onClick()

```
binding.fabBtnCreateNote.setOnClickListener {
    replaceFragment(CreateNoteFragment.newInstance(),false)
}
```

```
 private val onClicked = object :
 NotesAdapter.OnItemClickListener{
     override fun onClicked(noteId: Int) {


         val fragment :Fragment
         val bundle = Bundle()
         bundle.putInt("noteId",noteId)
         fragment =
 CreateNoteFragment.newInstance()
         fragment.arguments = bundle

         replaceFragment(fragment,false)
     }

 }
```

Gestori di eventi

- Esistono altri metodi che non fanno parte della classe View ma che possono influire direttamente sul modo di gestire gli eventi.
- `Activity.dispatchTouchEvent(MotionEvent)`
  Consente all'Activity di intercettare tutti gli eventi di tocco prima che vengano inviati alla finestra.
- `ViewGroup.onInterceptTouchEvent(MotionEvent)`
  Consente a ViewGroup di guardare gli eventi mentre vengono inviati alle viste figlio.
- `ViewParent.requestDisallowInterceptTouchEvent(boolean)`
  Chiamato su una vista genitore indica che non deve intercettare eventi di tocco con `onInterceptTouchEvent(MotionEvent)`.

Modalità touch

- Quando un utente sceglie di navigare un'interfaccia sfruttando i tasti direzionali o una trackball, è necessario mettere a fuoco gli elementi attivabili (come i pulsanti) in modo che l'utente possa vedere cosa accetterà l'input.
- Viceversa, se il dispositivo ha funzionalità di tocco e l'utente inizia ad interagire con l'interfaccia toccandola, non è più necessario evidenziare gli elementi o mettere a fuoco una particolare View.
- La modalità di interazione sopra descritta prende il nome di modalità touch.
- In modalità touch, solo le View per le quali `isFocusableInTouchMode()` è `true` saranno focalizzabili, come i widget di modifica del testo.

Modalità touch - 2

- Ogni volta che un utente utilizza un tasto direzionale o scorre con una trackball, il dispositivo uscirà dalla modalità touch e troverà una View da mettere a fuoco.
- Lo stato della modalità touch viene mantenuto in tutto il sistema (tutte le finestre e le attività).
- Per interrogare lo stato corrente, è possibile chiamare `isInTouchMode()` per vedere se il dispositivo è attualmente in modalità touch.

Gestione della messa a fuoco

- Il framework gestisce il movimento di messa a fuoco di routine in risposta all'input dell'utente. Ciò include il cambio di messa a fuoco quando le View vengono rimosse o nascoste, o quando nuove View diventano disponibili.
- Le View indicano il loro stato di messa a fuoco tramite il metodo `isFocusable()`, per apportare modificare si chiama il metodo `setFocusable()`.
- In modalità touch, è possibile richiedere se una View consente la messa a fuoco con `isFocusableInTouchMode()` e si può cambiare con `setFocusableInTouchMode()`.

Gestione della messa a fuoco - 2

- Il movimento della messa a fuoco si basa su un algoritmo che trova l'elemento più vicino in una determinata direzione.
- In rari casi, l'algoritmo predefinito potrebbe non corrispondere al comportamento previsto dallo sviluppatore.
- In queste situazioni, è possibile fornire override espliciti con i seguenti attributi XML nel file di layout:
  - `nextFocusDown`
  - `nextFocusLeft`
  - `nextFocusRight`
  - `nextFocusUp`

Gestione della messa a fuoco - 3

- Se si desidera dichiarare una View come attivabile nell'interfaccia utente (quando tradizionalmente non lo è), si può aggiungere alla View l'attributo XML `android:focusable` nella dichiarazione del layout, impostando il valore `true`.
- Si può anche dichiarare una View come attivabile in modalità touch usando `android:focusableInTouchMode`.
- Per richiedere che una View particolare sia messa a fuoco, chiamare `requestFocus()`.
- Per ricevere una notifica quando una View riceve o perde lo stato di focus attivo, utilizzare `onFocusChange()`.

Esempio android: focusable

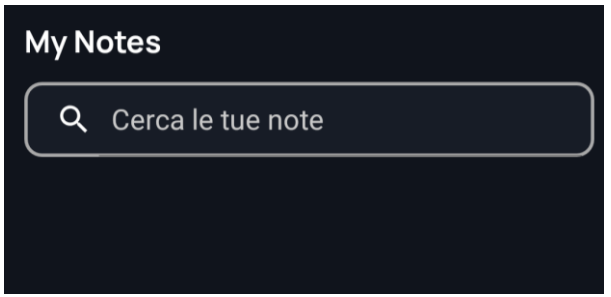
```
<?xml version="1.0" encoding="utf-8"?>
<selector
xmlns:android="http://schemas.android.com/apk/res
/android">

    <item android:state_enabled="true"
android:state_focused="true">
        <shape android:shape="rectangle">
<solid android:color="@color/colorPrimary"/>
            <corners android:radius="10dp"/>
            <stroke
android:color="@color/ColorAzure"
android:width="1dp"/>
        </shape>
    </item>

    <item android:state_enabled="true">
        <shape android:shape="rectangle">
            <solid
android:color="@color/colorPrimary"/>
            <corners android:radius="10dp"/>
            <stroke
android:color="@android:color/darker_gray"
android:width="1dp"/>
        </shape>
    </item>
</selector>
```

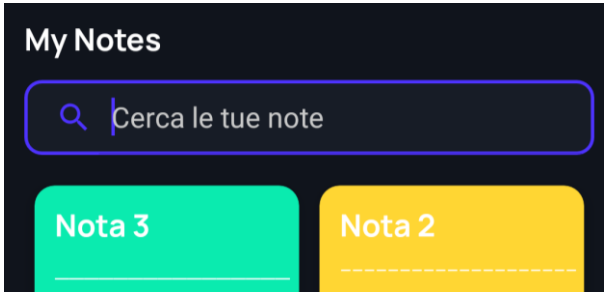
```
<EditText
    android:id="@+id/etNoteTitle"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="@dimen/_10sdp"
    android:background="@drawable/custom_edit_title"
    android:focusable="true"
    android:ellipsize="end"
    android:fontFamily="@font/manrope_bold"
    android:hint="@string/notes_title"
    android:inputType="text"
    android:maxLines="1"
    android:padding="@dimen/_10sdp"
    android:textColor="@color/ColorWhite"
    android:textColorHint="@color/ColorGray"
    android:textSize="@dimen/_14ssp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/imgBack"
    tools:ignore="Autofill" />
```

Esempio onFocusChance

```
binding.searchView.setOnQueryTextFocusChangeListener { _,
hasFocus ->
    binding.searchView.isSelected = hasFocus
    binding.searchView.isIconified = !hasFocus
}
```

```
  <selector
  xmlns:android="http://schemas.android.com/apk/res/android">
      <item android:state_selected="false">
          <shape android:shape="rectangle">
              <solid android:color="@color/colorPrimary"/>
              <corners android:radius="10dp"/>
              <stroke
  android:color="@android:color/darker_gray"
  android:width="2dp"/>
          </shape>
      </item>
      <item android:state_selected="true">
          <shape android:shape="rectangle">
              <solid android:color="@color/colorPrimary"/>
              <corners android:radius="10dp"/>
              <stroke android:color="@color/colorAccent"
  android:width="2dp"/>
          </shape>
      </item>
  </selector>
```





Utilizzo dei gesti tattili

- Android fornisce una varietà di API per creare e rilevare i gesti.
- L’aggiunta all’app di interazioni basate sul tocco, in molti casi, aumenta notevolmente l’utilità e l’attrattività della stessa.
- Per fornire agli utenti un'esperienza coerente e intuitiva, l’app deve seguire le convenzioni Android accettate per i gesti tattili.
- Quindi porre particolare attenzione su:
  - Gestione dei gesti di input
  - Creazione di un movimento fisicamente plausibile
  - Rendere fluide le transizioni

Rilevare gesti

- Un `gesto tattile` si verifica quando l'utente posiziona una o più dita sul touch screen e l'applicazione interpreta tale motivo di tocco come un particolare gesto.
- Il rilevamento dei gesti si divide in due fasi:
  1. Raccogliere dati sugli eventi di tocco.
  2. Interpretare i dati per vedere se soddisfano i criteri per uno qualsiasi dei gesti supportati dall'app.
- Se l'app usa `gesti comuni` come doppio tocco, lunga pressione, lancio e così via, si può utilizzare la classe `GestureDetector`.
- Tale classe consente di rilevare facilmente i gesti comuni senza dover elaborare "a mano" i singoli eventi di tocco.

Rilevare gesti - 2

- Alcuni dei gesti che la classe `GestureDetector` supporta sono:
  - `onDown()`
  - `onLongPress()`
  - `onFling()`
  - `...`
- È possibile utilizzare `GestureDetector` insieme al metodo `onTouchEvent()`.
- Quando si crea un'istanza di un oggetto `GestureDetectorCompat`, uno dei parametri necessari è una classe che implementa l'interfaccia `GestureDetector.OnGestureListener`,
  la quale notifica agli utenti quando si è verificato un particolare evento di tocco.
- Se si desidera elaborare solo alcuni, è possibile estendere `GestureDetector.SimpleOnGestureListener` anziché implementare l'interfaccia `GestureDetector.OnGestureListener`.
- In questo modo si può fare l'override dei soli metodi che interessano.

Tenere traccia dei movimenti del tocco e del puntatore

- Un nuovo `onTouchEvent()` viene attivato con un evento `ACTION_MOVE` ogni volta che la posizione, la pressione o le dimensioni del contatto corrente cambiano.
- Tutti gli eventi vengono registrati nel `MotionEvent` parametro di `onTouchEvent()`.
- Poiché il tocco con le dita non è sempre la forma più precisa di interazione, il rilevamento degli eventi di tocco è spesso basato più sul movimento che sul semplice contatto.
- Per aiutare le app a distinguere tra gesti basati sul movimento (come uno scorrimento) e non (come un singolo tocco), Android include la nozione di "touch slop".

Touch Slop e Velocità della traccia

- Lo "slop" al tocco si riferisce alla distanza in pixel che il tocco di un utente può percorrere prima che il gesto venga interpretato come un gesto basato sul movimento.
- Esistono diversi modi per tenere traccia del movimento in un gesto.
  Per esempio:
  - La posizione iniziale e finale di un puntatore;
  - La direzione in cui sta viaggiando il puntatore determinata dalle coordinate X e Y;
  - Utilizzando `getHistorySize()` che restituisce il numero di punti, ovvero di movimenti che si sono verificati tra l'evento corrente e il precedente;
  - La velocità del puntatore mentre si sposta sul touch screen.
- Per semplificare il calcolo della velocità, Android fornisce la classe `VelocityTracker` che aiuta a monitorare la velocità degli eventi di tocco.
- Nota: Calcolare la velocità dopo un evento `ACTION_MOVE` e non dopo un evento `ACTION_UP` dove le velocità X e Y saranno 0.

Acquisizione e rilascio del puntatore

- Una View può richiedere l'acquisizione del puntatore solo quando è presente un'azione utente specifica su di essa. La richiesta avviene chiamando il metodo `requestPointerCapture()`.
- Una volta che la View ha acquisito il puntatore con successo, Android inizia a fornire gli eventi del mouse alla View focalizzata, la quale può gestire tali eventi eseguendo una delle seguenti attività:
  - L'override `onCapturedPointerEvent(MotionEvent)`, se si vuole utilizzare una View personalizzata
  - `onCapturedPointerListener`, altrimenti.
- In entrambi i casi, la View riceve un `MotionEvent` con le coordinate del puntatore. È possibile recuperare le coordinate utilizzando `getX()` e `getY()`.
- Il rilascio dell'acquisizione del puntatore può avvenire in due modi:
  - chiamata esplicita di `releasePointerCapture()` da parte della View.
  - rilascio automatico da parte del sistema quando la gerarchia di visualizzazione che contiene la View che aveva richiesto l'acquisizione perde lo stato focus.

Animare un gesto di scorrimento

- In Android, lo scorrimento si ottiene utilizzando la `ScrollView` e quindi gli scroller (`Scroller` o `OverScroller`) per raccogliere i dati necessari a produrre un'animazione di scorrimento in risposta ad un evento di tocco.
- Lo scorrimento è il processo generale di spostamento della finestra di visualizzazione. Quando lo scorrimento è su entrambi gli assi X e Y, viene detto *panning* (panoramica).
- Il *trascinamento* è il tipico scorrimento che si verifica quando un utente trascina il dito sul touch screen. Tale processo può essere implementato facendo l'override di `onScroll()` in `GestureDetector.OnGestureListener`.
- Il *flinging* (lancio) è il tipo di scorrimento che si verifica quando un utente trascina e solleva rapidamente il dito. Tale processo può essere implementato facendo l'override di `onFling()` in `GestureDetector.OnGestureListener` e utilizzando un oggetto scroller.

Gesti multi-touch

- Un gesto multi-touch avviene quando più puntatori (dita) toccano lo schermo contemporaneamente.
- Il sistema genera i seguenti eventi di tocco:
  - `ACTION_DOWN`
  - `ACTION_POINTER_DOWN`
  - `ACTION_MOVE`
  - `ACTION_POINTER_UP`
  - `ACTION_UP`

Gesti multi-touch - 2

- Si tiene traccia dei singoli puntatori all'interno di un `MotionEvent` tramite l'indice e l'ID di ogni puntatore:
  - Un MotionEvent memorizza in un array le informazioni su ogni puntatore. L'indice di un puntatore è la sua posizione all'interno di questo array e si ottiene chiamando il metodo `findPointerIndex()`.
  - Ogni puntatore ha anche un ID che rimane persistente tra gli eventi di tocco (finché il puntatore è attivo) per consentire il rilevamento di un singolo puntatore sull'intero gesto. L'ID si ottiene con il metodo `getPointerID()`.
- Per recuperare l'azione di un file `MotionEvent` usare il metodo `getActionMasked()`, progettato per funzionare con più puntatori. Tale metodo restituisce l'azione mascherata eseguita senza includere i bit dell'indice del puntatore. Per recuperare l'indice del puntatore associato all'azione usare il metodo `getActionIndex()`.

Trascinare e ridimensionare

- In un'operazione di trascinamento (o scorrimento), l'app deve distinguere tra puntatore (dito) originale e gli altri puntatori successivi (eventuali altre dita posizionate sullo schermo).
- A tale scopo l'app tiene traccia degli eventi `ACTION_POINTER_DOWN` e `ACTION_POINTER_UP`, che vengono passati al callback `onTouchEvent()` ogni volta che un puntatore secondario va verso il basso o verso l'alto.
- Per operazioni di ridimensionamento Android fornisce `ScaleGestureDetector`, che può essere utilizzato insieme a `GestureDetector` quando si desidera che una View riconosca gesti aggiuntivi.
- Per segnalare il rilevamento di eventi di gesto, `ScaleGestureDetector` utilizza `ScaleGestureDetector.OnScaleGestureListener`.

Intercettare eventi di tocco

- onInterceptTouchEvent() viene chiamato ogni volta che viene rilevato un touch sulla superficie di un oggetto ViewGroup
  - se onInterceptTouchEvent() ritorna true l'oggetto MotionEvent viene intercettato e passato al metodo onTouchEvent() del padre. La visualizzazione figlio che gestiva gli eventi di tocco riceve un ACTION_CANCEL.
  - se onInterceptTouchEvent() ritorna false e spia gli eventi fino ai loro obiettivi abituali che gestiranno gli eventi con i propri onTouchEvent().
- ViewGroup fornisce un metodo requestDisallowInterceptTouchEvent() che viene chiamato quando un figlio non desidera che l'elemento padre e i predecessori intercettino gli eventi di tocco con onInterceptTouchEvent().
- Se una ViewGroup riceve un MotionEvent con un ACTION_OUTSIDE l'evento non verrà inviato ai figli per impostazione predefinita. (eseguire override del metodo dispatchTouchEvent(MotionEvent())
- La classe TouchDelegate consente a un genitore di estendere l'area toccabile di una view figlio oltre i limiti della view figlio.

Esempio

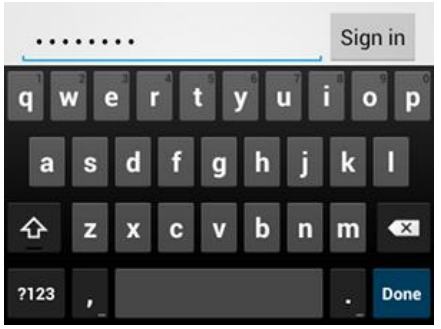
```
class MyViewGroup @JvmOverloads constructor(
    context: Context,
    private val mTouchSlop: Int = ViewConfiguration.get(context).scaledTouchSlop
) : ViewGroup(context) {

  ...

  override fun onInterceptTouchEvent(ev: MotionEvent): Boolean {
    /*
     * This method JUST determines whether we want to intercept the motion.
     * If we return true, onTouchEvent will be called and we do the actual
     * scrolling there.
     */
    return when (ev.actionMasked) {
      MotionEvent.ACTION_CANCEL, MotionEvent.ACTION_UP -> {
        mIsScrolling = false
      }
      MotionEvent.ACTION_MOVE -> {
        if (mIsScrolling) {
          true
        } else {
          val xDiff: Int = calculateDistanceX(ev)

          if (xDiff > mTouchSlop) {
            mIsScrolling = true
            true
          } else {
            false
          }
        }
      }
      ...
      else -> {
        false
      }
    }
  }

  override fun onTouchEvent(event: MotionEvent): Boolean {
    // Here we actually handle the touch event (e.g. if the action is ACTION_MOVE,
    // scroll this container).
    // This method will only be called if the touch event was intercepted in
    // onInterceptTouchEvent
    ...
  }
}
```

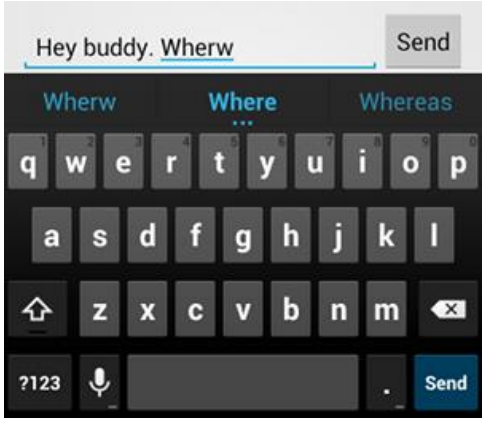
Tipo di tastiera

- Per dichiarare il metodo di input per i campi di testo bisogna aggiungere l'attributo android:inputType all'elemento <EditText>.
  - "phone": metodo di input per l'immissione di un numero di telefono
  - "textPassword": metodo di input per l'immissione di password (nasconde input)
  - "textAutoCorrect": per abilitare correzione automatica
  - "textMultiLine": per avere il testo su più linee

"phone"

"textPassword"

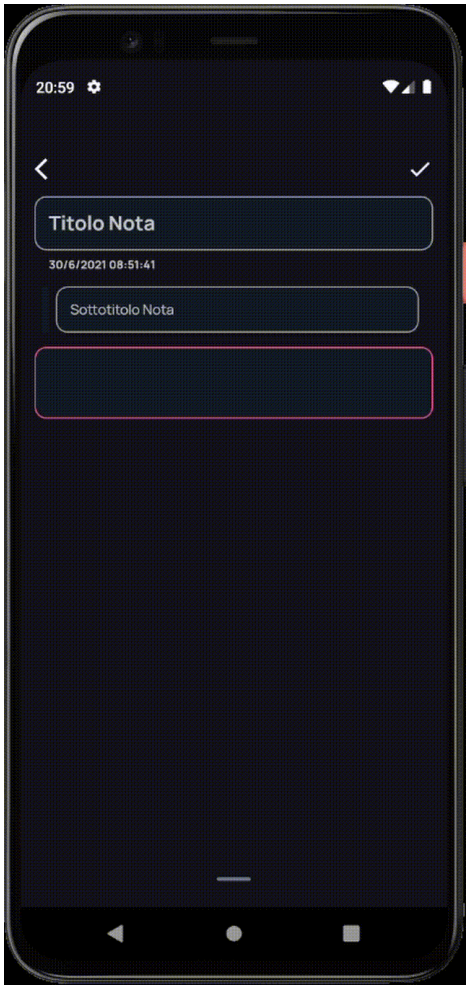




"textAutoCorrect"

Esempio textMultiLine

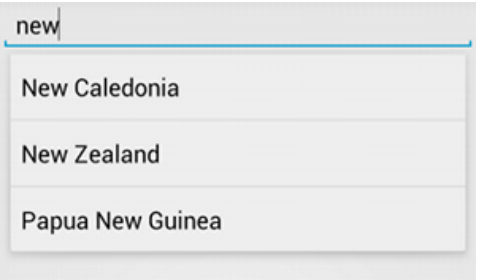
```
<EditText
    android:id="@+id/etNoteDesc"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="@dimen/_10sdp"
    android:background="@drawable/custom_edit_text"
    android:focusable="true"
    android:fontFamily="@font/manrope_regular"
    android:hint="@string/notes_desc"
    android:inputType="textMultiLine"
    android:padding="@dimen/_10sdp"
    android:textColor="@color/ColorWhite"
    android:textColorHint="@color/ColorGray"
    android:textSize="@dimen/_12ssp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/tvWebLink"
    tools:ignore="Autofill" />
```



Tipo di tastiera (2)

- È possibile combinare comportamenti e stili di metodi di input diversi
- Per specificare il pulsante di azione della tastiera utilizzare l'attributo android:imeOptions
  - "actionSend": viene visualizzato il pulsante invia
  - "actionSearch": viene visualizzato il pulsante cerca

- È possibile ascoltare le pressioni sul pulsante di azione definendo un TextView.OnEditorActionListener per l'elemento EditText
- Per fornire suggerimenti all'utente utilizzare la sottoclasse di EditText denominata AutoCompleteTextView e specificare un adapter che fornisca i suggerimenti di testo



Visibilità metodo di input

- Android mostra e nasconde in maniera automatica il metodo di input quando lo stato attivo dell'input si sposta all'interno o all'uscita da un campo di testo modificabile
- Si può controllare direttamente la visibilità del metodo di input
- Per visualizzare il metodo di input all'avvio dell'attività aggiungere l'attributo android:windowSoftInputMode all'elemento <activity> con valore "stateVisible"
- Per assicurarsi che il metodo di input sia visibile è possibile utilizzare InputMethodManager
- Quando viene mostrato riduce la visibilità dell'interfaccia, il trattamento preferito può essere indicato con l'attributo android:windowSoftInputMode nell'elemento <activity> del manifesto con uno dei valori "adjust"

Esempio

```
<activity
    android:name=".MainActivity"
    android:windowSoftInputMode="stateHidden|adjustResize" />
```

Esplorazione della tastiera

- Quando un utente naviga nell'app usando il tasto TAB della tastiera il sistema passa lo stato attivo di input tra gli elementi in base all'ordine in cui vengono visualizzati nel layout
- Per definire in modo esplicito l'ordine di messa a fuoco si usa l'attributo android:nextFocusForward
- Si può specificare anche l'ordine di messa a fuoco in una determinata direzione in modo esplicito utilizzando gli attributi:
  - Android:nextFocusUp
  - Android:nextFocusDown
  - Android:nextFocusLeft
  - Android:nextFocusRight

Input da tastiera

- Se si desidera intercettare o gestire direttamente l'input della tastiera è possibile farlo implementando metodi di callback dell'interfaccia KeyEvent.Callback
- Le classi Activity e View implementano l'interfaccia KeyEvent.Callback quindi in genere è necessario eseguire l'override dei metodi di callback nell'estensione di queste classi in base alle esigenze
- Per gestire l'evento di pressione del tasto implementare OnKeyUp() o OnKeyDown() in modo appropriato
- Quando un tasto viene combinato con i tasti di modifica (es MAIUSC o CTRL) è possibil eseguire una query KeyEvent sul tasto passato al metodo di callback
- Esistono diversi metodi per la gestione dei tasti di modifica (getModifiers() e getMetaState()) ma la soluzione più semplice è controllare se il tasto di modifica di interesse isShiftPressed() e isCtrlPressed()

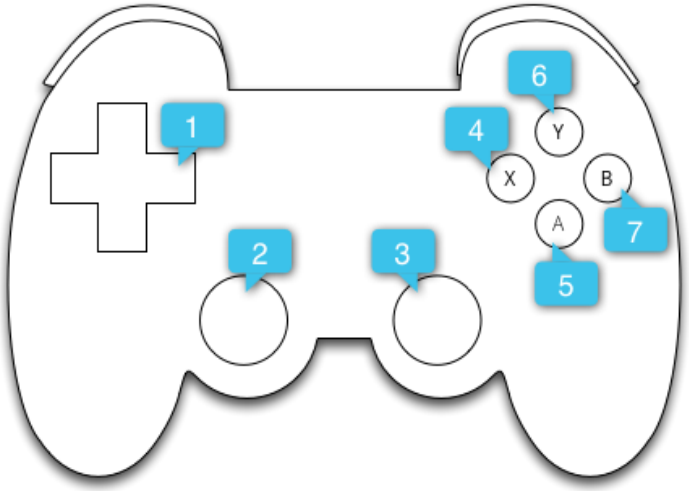
Esempio

```
override fun onKeyUp(keyCode: Int, event: KeyEvent): Boolean {
    return when (keyCode) {
        ...
        KeyEvent.KEYCODE_J -> {
            if (event.isShiftPressed) {
                fireLaser()
            } else {
                fireMachineGun()
            }
            true
        }
        KeyEvent.KEYCODE_K -> {
            if (event.isShiftPressed) {
                fireSeekingMissle()
            } else {
                fireMissile()
            }
            true
        }
        else -> super.onKeyUp(keyCode, event)
    }
}
```

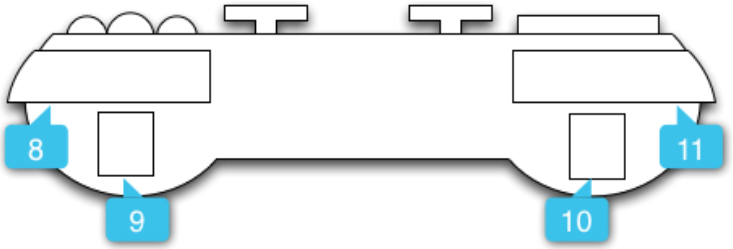
Gestire le azioni del controller

- Quando un controller si accoppia a un dispositivo android il sistema rileva automaticamente il controller come dispositivo di input e inizia a segnalare i suoi eventi di input
- Il gioco può ricevere eventi di input implementando i metodi di callback nell'activity o nella view (non in entrambi)
- L'approccio consigliato è quello di gestire gli eventi dalla view specifica con cui l'utente interagisce. (KeyEvent descrive gli eventi dei pulsanti, MotionEvent descrive l'input dei movimenti del joystick e delle levette)
- Per verificare che un dispositivo di input connesso sia un controller di gioco chiamare getSources() per ottenere un campo di bit combinato di tipi di origine supportati su esso
- Per giochi con più controller è possibile assegnare l'ID di ogni controller a ciascun giocatore

Front View



Top View



1. AXIS_HAT_X, , , , ,AXIS_HAT_YDPAD_UPDPAD_DOWNDPAD_LEFTDPAD_RIGHT
2. AXIS_X, , AXIS_YBUTTON_THUMBL
3. AXIS_Z, , AXIS_RZBUTTON_THUMBR
4. BUTTON_X
5. BUTTON_A
6. BUTTON_Y
7. BUTTON_B
8. BUTTON_R1
9. AXIS_RTRIGGER, AXIS_THROTTLE
10. AXIS_LTRIGGER, AXIS_BRAKE
11. BUTTON_L1

Esempio

- **Elaborazione pressione pulsanti**

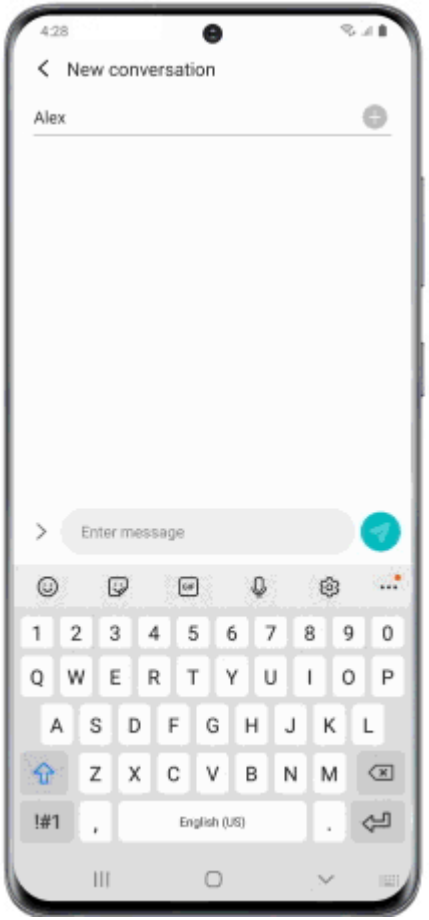
```
class GameView(...) : View(...) {
   ...

   override fun onKeyDown(keyCode: Int, event: KeyEvent): Boolean {
      var handled = false
      if (event.source and InputDevice.SOURCE_GAMEPAD ==
InputDevice.SOURCE_GAMEPAD) {
         if (event.repeatCount == 0) {
            when (keyCode) {
               // Handle gamepad and D-pad button presses to navigate the ship
               ...

               else -> {
                  keyCode.takeIf { isFireKey(it) }?.run {
                     // Update the ship object to fire lasers
                     ...
                     handled = true
                  }
               }
            }
         }
         if (handled) {
            return true
         }
      }
      return super.onKeyDown(keyCode, event)
   }
   // Here we treat Button_A and DPAD_CENTER as the primary action
   // keys for the game.
   private fun isFireKey(keyCode: Int): Boolean =
         keyCode == KeyEvent.KEYCODE_DPAD_CENTER || keyCode ==
KeyEvent.KEYCODE_BUTTON_A
}
```

- **Elaborazione movimenti del joystick**

```
private fun processJoystickInput(event: MotionEvent, historyPos: Int) {

   val inputDevice = event.device

   // Calculate the horizontal distance to move by
   // using the input value from one of these physical controls:
   // the left control stick, hat axis, or the right control stick.
   var x: Float = getCenteredAxis(event, inputDevice,
MotionEvent.AXIS_X, historyPos)
   if (x == 0f) {
      x = getCenteredAxis(event, inputDevice,
MotionEvent.AXIS_HAT_X, historyPos)
   }
   if (x == 0f) {
      x = getCenteredAxis(event, inputDevice, MotionEvent.AXIS_Z,
historyPos)
   }

   // Calculate the vertical distance to move by
   // using the input value from one of these physical controls:
   // the left control stick, hat switch, or the right control stick.
   var y: Float = getCenteredAxis(event, inputDevice,
MotionEvent.AXIS_Y, historyPos)
   if (y == 0f) {
      y = getCenteredAxis(event, inputDevice,
MotionEvent.AXIS_HAT_Y, historyPos)
   }
   if (y == 0f) {
      y = getCenteredAxis(event, inputDevice, MotionEvent.AXIS_RZ,
historyPos)
   }

   // Update the ship object based on the new x and y values
}
```

Metodo di Input

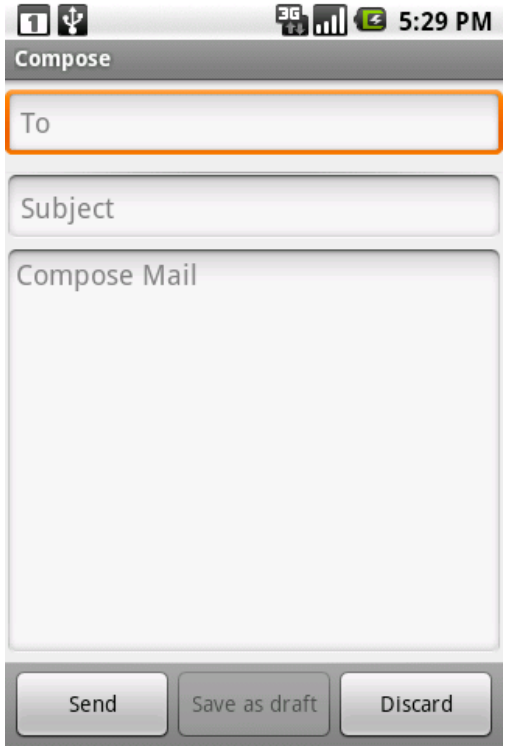
- Che cos'è un metodo di input?
  - Un metodo di input è uno strumento in un sistema operativo o in un programma software che consente all'utente di un dispositivo informatico di utilizzare la tastiera per digitare caratteri che non sono rappresentati sulla tastiera di quel particolare dispositivo.



- Android fornisce un framework di input-method estensibile che consente alle applicazioni di fornire agli utenti metodi di input alternativi, come tastiere su schermo o persino input vocale.
- Per aggiungere un IME al sistema Android, è necessario creare un'applicazione Android contenente una classe che estende il metodo.

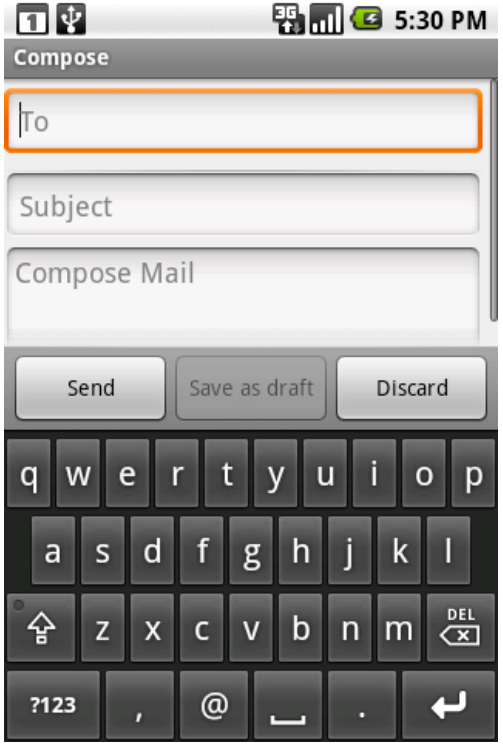
Metodi di Input su schermo

- A partire da Android 1.5, la piattaforma Android offre un Framework del metodo di input (IMF) che consente di creare metodi di input sullo schermo come le tastiere software.
- Android IMF è progettato per supportare una varietà di IMI, tra cui tastiera morbida, riconoscimento della scrittura a mano e traduttori hard keyboard. Il nostro focus, tuttavia, sarà sulle tastiere morbide, poiché questo è il tipo di metodo di input che attualmente fa parte della piattaforma.
- Un utente accede in genere all'IME corrente toccando una visualizzazione di testo da modificare, come mostrato qui nella schermata iniziale:

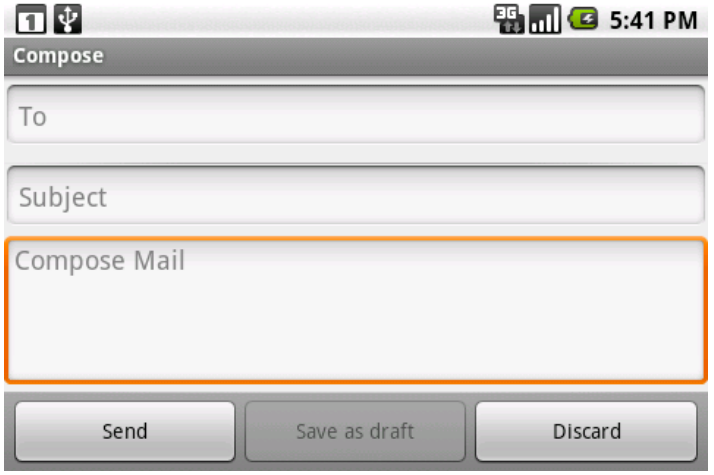


Un utente accede in genere all'IME corrente toccando una visualizzazione di testo da modificare, come nell'esempio.

L'approccio usato è chiamato *pan and scan*e. Che comporta semplicemente lo scorrimento della finestra dell'applicazione in modo che la visualizzazione attualmente focalizzata sia visibile.

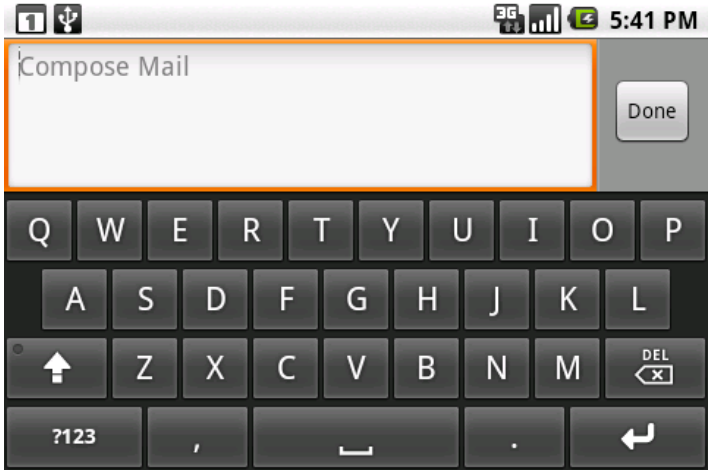


Le dimensioni della finestra dell'applicazione vengono modificate in modo che nessuna di essa sia nascosta dall'IME, consentendo l'accesso completo sia all'applicazione che all'IME.



In questo caso la finestra dell'applicazione viene lasciata così com'è e l'IME visualizza semplicemente lo schermo intero sopra di esso, come mostrato qui.

L'ultima modalità principale è *la modalità a schermo* intero o *di* estrazione.



Ciclo di vita dell'IME

Il diagramma seguente descrive il ciclo di vita di un IME

Design del Metodo di Input UI

Ci sono due elementi visivi principali per progettare l'interfaccia del metodo di input: l'input view e il candidates view.

Input View
L'input view è un UI in cui l'utente immette testo sotto forma di click con tasti o gesti. Quando l'IME viene visualizzato per la prima volta, il sistema chiama la callback CreateInputView().

```
override fun onCreateInputView(): View {
    return layoutInflater.inflate(R.layout.inpu
t, null).apply {
        if (this is MyKeyboardView) {
            setOnKeyboardActionListener(this@MyI
nputMethod)
            keyboard = latinKeyboard
        }
    }
}
```

Candidates View
Il candidates view è l'interfaccia utente in cui l'IME visualizza potenziali correzioni di parole o suggerimenti da selezionare per l'utente. Nel ciclo di vita dell'IME, il sistema chiama quando è pronto per visualizzare la candidates view.

Testo intorno al cursore

Quando si gestisce la modifica del testo esistente in un campo di testo, alcuni dei metodi più utili in sono:

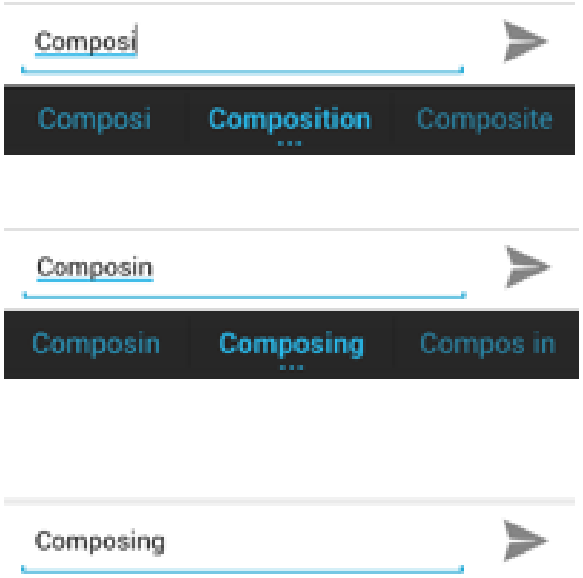
- **GetTextBeforeCursor**()
  - Restituisce un oggetto contenente il numero di caratteri richiesti prima della posizione corrente del cursore.
- **GetTextAfterCursor**()
  - Restituisce un oggetto contenente il numero di caratteri richiesti dopo la posizione corrente del cursore.
- **DeleteSurroundingText**()
  - Elimina il numero specificato di caratteri prima e dopo la posizione corrente del cursore.
- **CommitText**()
  - Eseguire il commit di un oggetto nel campo di testo e impostare una nuova posizione del cursore.

Ad esempio, il frammento seguente mostra come sostituire i quattro caratteri a sinistra del cursore con il testo "Hello!":

```
currentInputConnection.also { ic:
InputConnection ->
    ic.deleteSurroundingText(4, 0)
    ic.commitText("Hello", 1)
    ic.commitText("!", 1)
}
```

Previsione del testo

Se l'IME eseguisse la previsione del testo è possibile visualizzare lo stato di avanzamento nel campo di testo fino a quando l'utente non esegue il commit della parola, quindi è possibile sostituire la composizione parziale con il testo completato.



Il frammento seguente mostra come mostrare lo stato di avanzamento in un campo di testo:

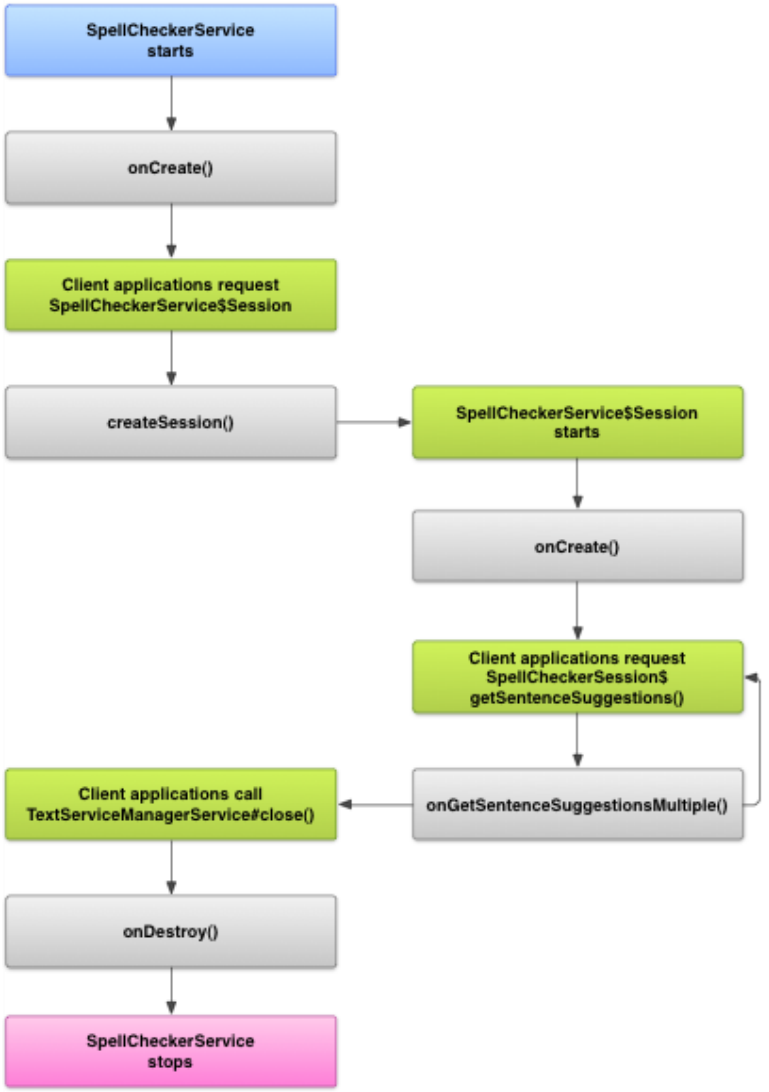
```
currentInputConnection.also { ic: InputConnection
->
    ic.setComposingText("Composi", 1)
    ic.setComposingText("Composin", 1)
    ic.commitText("Composing ", 1)
}
```

Framework di controllo ortografico

- La piattaforma Android offre un framework di controllo ortografico che ti consente di implementare e accedere al controllo ortografico nella tua applicazione.

- Il framework è una delle API del servizio di testo offerte dalla piattaforma Android.

- Per usare il framework nella tua app, crei un tipo speciale di servizio Android che genera un oggetto sessione del correttore ortografico. In base al testo fornito, l'oggetto sessione restituisce suggerimenti ortografici generati dal correttore ortografico.



Il diagramma seguente mostra il ciclo di vita del servizio correttore ortografico.

Per avviare il controllo ortografico vengono seguiti alcuni step:

1. l'app avvia l'implementazione del servizio di controllo ortografico.

2. I client dell'app, ad esempio attività o singoli elementi dell'interfaccia utente, richiedono una sessione di controllo ortografico al servizio, quindi usano la sessione per ottenere suggerimenti per il testo.

3. Quando un client termina il funzionamento, chiude la sessione del correttore ortografico. Se necessario, l'app può arrestare il servizio di controllo ortografico in qualsiasi momento.

- Per usare il framework nell'app bisogna definire alcune classi di controllo grafico:

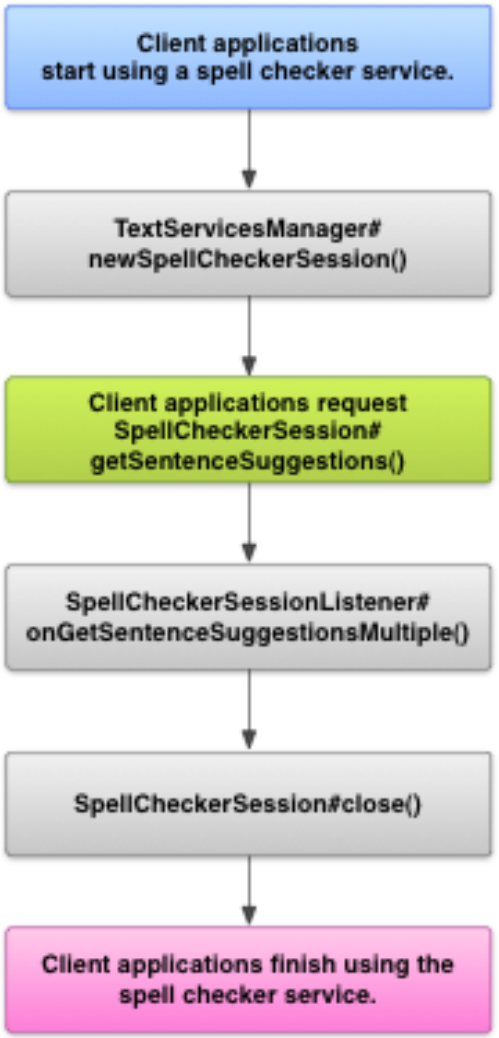
- **Una sottoclasse di SpellCheckerService**: Implementa sia la classe che l'interfaccia del framework del correttore ortografico. All'interno della sottoclasse è necessario implementare il metodo seguente:
  - **CreateSession**() : Metodo factory che restituisce un oggetto a un client che desidera eseguire il controllo ortografico: SpellCheckerService.Session.
- **L'attuazione di SpellCheckerService.Session** : Oggetto fornito dal servizio di controllo ortografico ai client, che consente loro di passare testo al correttore ortografico e ricevere suggerimenti. All'interno di questa classe, è necessario implementare i seguenti metodi:
  - **OnCreate**() : In questo metodo è possibile inizializzare l'oggetto in base alle impostazioni locali correnti: createSession()
  - **OnGetSentenceSuggestionsMultiple**() :Fa il controllo ortografico effettivo. Questo metodo restituisce una matrice di suggerimenti contenenti per le frasi passate. SentenceSuggestionsInfo.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.samplespellcheckerservice" >
    <application
        android:label="@string/app_name" >
        <service
            android:label="@string/app_name"
            android:name=".SampleSpellCheckerService"
            android:permission="android.permission.BIND_TEXT_SERVICE" >
            <intent-filter >
                <action
android:name="android.service.textservice.SpellCheckerService" />
            </intent-filter>

            <meta-data
                android:name="android.view.textservice.scs"
                android:resource="@xml/spellchecker" />
        </service>

        <activity
            android:label="@string/sample_settings"
            android:name="SpellCheckerSettingsActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

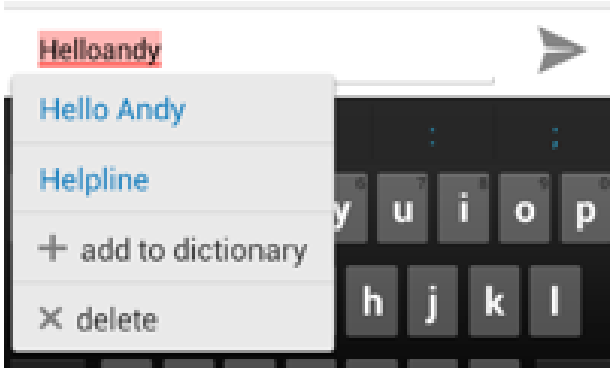
Oltre al codice, è necessario fornire il file manifesto appropriato e un file di metadati per il correttore ortografico.

Il file manifesto definisce l'applicazione, il servizio e l'attività per il controllo delle impostazioni.

Accedere al servizio di controllo ortografico da un client



Le applicazioni che usano le visualizzazioni beneficiano automaticamente del controllo ortografico, perché usa automaticamente un correttore ortografico.



Tuttavia, potresti voler interagire direttamente con un servizio di controllo ortografico anche in altri casi. Il diagramma seguente mostra il flusso di controllo per l'interazione con un servizio di controllo ortografico.