Anotaciones Click, Touch, LongClick y AfterTextChange

Click

Este evento detecta únicamente cuando el usuario hace un clic sobre un objeto de UI. Hay 4 diferentes maneras de implementar este evento:

https://github.com/androidannotations/androidannotations/wiki/ClickEvents#click

- Anotación con parámetro de entrada (es decir, especificando el recurso):

```
@Click(R.id.b_dot)
void buttonDotClicked()
```

- Anotación sin parámetro de entrada: El nombre del método Java será la referencia:

```
@Click
void b_equal()
```

- Método con una View como parámetro de entrada: El evento View.OnClickListener permite aceptar una View como parámetro de entrada para poder operar sobre el objeto que ha desencadenado el evento.

```
@Click
void yetAnotherButton(View clickedView)
```

- Anotación con múltiples parámetros de entrada: Para cuando quieras ejecutar el mismo evento sobre múltiples objetos:

```
@Click({R.id.b_0, R.id.b_1, R.id.b_2, R.id.b_3, R.id.b_4, R.id.b
void bDigitalButtons()
```

Además se pueden mezclar las distintas metodologías vistas anteriormente, por ejemplo "Anotación con múltiples parámetros de entrada" y "Método con una View como parámetro de entrada":

```
@Click({R.id.b_0, R.id.b_1, R.id.b_2, R.id.b_3, R.id.b_4, R.id.b
void bDigitalButtons(View clickedView)
```

Touch

Es similar a la anotación "@Click" pero permite el uso de del evento "MotionEvent" como parámetro de entrada al método para poder obtener más funcionalidades.

https://github.com/androidannotations/androidannotations/wiki/ClickEvents#other-events

Tienes las siguientes opciones:

- Anotación con parámetro de entrada, es decir, especificando el recurso
- Anotación sin parámetro de entrada: El nombre del método Java será la referencia
- Método con una View como parámetro de entrada
- Anotación con múltiples parámetros de entrada
- View & MotionEvent como parámetros de entrada al método:

```
@Touch
void bOperationsButtons(View v, MotionEvent event)
```

- Method inputs parameter (MotionEvent)

```
@Touch
void bOperationsButtons(MotionEvent event)
```

Click vs Touch

Click: https://developer.android.com/reference/android/view/View.OnClickListener

Touch: https://developer.android.com/reference/android/view/View.OnTouchListener

El evento `onTouch` da la capacidad de tener el "Motion Event". Con esta funcionalidad podrás hacer y mejorar los metodos realizados por ejemplo a separar los estados del evento (en lo que se refiere al movimiento del dedo). Ejemplo:

```
MotionEvent.ACTION_UP
MotionEvent.ACTION_DOWN
MotionEvent.ACTION_MOVE
```

El evento `onClick` permite saber con que objeto está interactuando el usuario. `onClick` da la funcionalidad completa y comprimida de los eventos de `focusing`, `pressing` y `releasing` por lo que se pierde esta funcionalidad, pero es su favor se gana simpleza.

LongClick

Es usado para detectar el evento de "long click". La documentación de AndroidAnnotations no da mucha más infromación:

https://github.com/androidannotations/androidannotations/wiki/ClickEvents#other-events

Pero podemos mirar la documentación oficial de Android:

https://developer.android.com/reference/android/view/View.OnLongClickListener

Como podemos ver "OnLongClickListener" es un método abstracto con un único parámetro de entrada (View v) exáctamente igual que el evento `OnClickListener` así que la anotación `@LongClick` tendrá la misma funcionalidad que la anotación `@Click` annotation.

Notas importantes

Es importante saber que si tienes la anotación `@Touch` sobre un objeto de UI entonces no podrás emplear la anotación `@LongClick`. Tendrás que hacer la detección del evento de "long click" manualmente (a través de la detección de tiempo).

Pseudocódigo:

```
if MotionEvent.ACTION_DOWN then
    var actionDown = getActualmoment
else if MotionEvent.ACTION_UP then
    var actionUp = getActualmoment
    if (actionUp-actionDown) > 1 sec then
        this is a LongClick event
```

Otra nota a tener en cuenta es que no podrás tener dos métodos de igual nombre con diferentes anotaciones, por ejemplo:

```
@Click
void b_del()
@LongClick
void b_del()
```

Si necesitas la anterior opción, puedes elaborar un"fix" usando el parámetro de entrada `View`:

```
@Click
void b_del(View v)
@LongClick
void b_del()
```

AfterTextChange

Ésta anotación hará que el método que tiene la anotación sea lanzado después que el texto haya cambiado.

https://github.com/androidannotations/androidannotations/wiki/TextChangeEvents#aftertextchange

Existen 4 manera diferentes de implementar esta anotación:

- Anotación con un parámetro de entrada y un parámetro de entrada al método que tiene la anotación:

```
@AfterTextChange(R.id.helloTextView)
void afterTextChangedOnHelloTextView(Editable text, TextView hell
```

android.widget.TextView: parámetro para saber qué vista ha generado este evento.

android.text.Editable: para hacer cambios en el texto modificado.

- Anotación sin un parámetro de entrada y un parámetro de entrada al método que tiene la anotación:

```
@AfterTextChange
void helloTextViewAfterTextChanged(TextView hello)
```

android.widget.TextView: parámetro para saber qué vista ha generado este evento.

- Anotación con múltiples parámetros de entrada y un parámetro de entrada al método que tiene la anotación:

```
@AfterTextChange({R.id.editText, R.id.helloTextView})
void afterTextChangedOnSomeTextViews(TextView tv, Editable text)
```

android.widget.TextView: parámetro para saber qué vista ha generado este evento.

android.text.Editable: para hacer cambios en el texto modificado.

- Anotación con un parámetro de entrada y sin parámetro de entrada al método que tiene la anotación:

```
@AfterTextChange(R.id.helloTextView)
void afterTextChangedOnHelloTextView()
```

Al igual que los casos anteriores también se pueden mezclas las diferentes opciones.