

Interfaz de usuario:

“Manejo de Eventos de Entrada”

Eventos de Entrada (I)

- ▶ El objetivo principal es recoger los eventos de un usuario cuando trabaja sobre una vista.
- ▶ Por ejemplo:
 - ☑ Pulsar sobre un botón, textview,... o cualquier objeto tipo view.
- ▶ Veremos que tenemos distintas posibilidades de hacerlo pero cada una de ellas tiene sus ventajas e inconvenientes

Eventos de Entrada (II)

- ▶ Teniendo en cuenta los distintos dispositivos android existentes, vemos que incorporan diferentes **mecanismos de interacción con el usuario**:
 - ☒ Pantalla táctil
 - ☒ Teclado
 - ☒ Trackball
 - ☒ Sensores
 - ☒ ...
- Independientemente del elemento de entrada utilizado que produzca un **evento de entrada**, estos se **capturan en nuestra aplicación android** y se tratan de la misma forma.
- Vamos a ver **varias técnicas alternativas para reaccionar ante estos eventos**.

Eventos de Entrada (III)

Captura de Eventos de Usuario

- ▶ Disponemos de **3 alternativas para capturar eventos** de usuario (*cada una de ellas tiene ventajas e inconvenientes*):
 - ⌘ Definir el **atributo o propiedad `onClick`** de la vista
 - *Muy sencillo, pero **sólo funciona con el evento `onClick`** (no por ejemplo, para `onLongClick`)*
 - ⌘ Usar **Escuchadores de eventos (*Event Listener*)**
 - *Es el método más **complejo**, pero **es general** (para cualquier tipo de evento desde cualquier clase).*
 - ⌘ Usar **Manejadores de eventos (*Event Handler*)**
 - *Sencillo, pero **sólo puede definirse dentro de una clase `View`** para atender eventos generados por la clase `View`.*

Eventos de Entrada (IV)

Captura de Eventos de Usuario: atributo onClick

- ▶ A partir de la **versión 1.6 de Android**, las vistas tienen el **atributo "onClick"**, donde **se puede indicar el nombre de un método que será invocado al hacer *click* en la vista.**
- ▶ En el **código xml** de la **vista**:

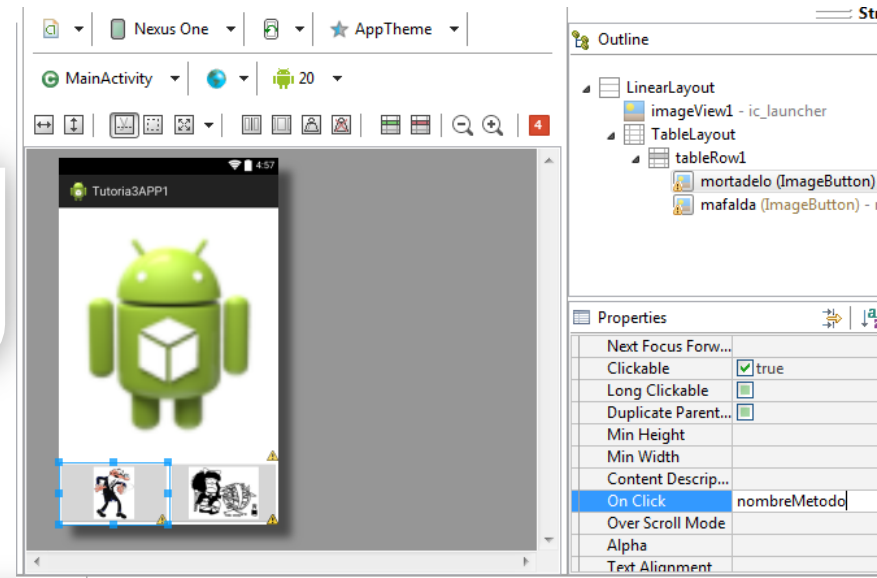
```
<Button ...  
    android:onClick="sePulsa"/>
```

A continuación, en la **activity** que **visualiza la vista** implementamos el **método** ("**sePulsa**" en nuestro **ejemplo**):

```
public void sePulsa(View view){  
    ...  
}
```

Es obligatorio

Sencillo y limpio, pero **sólo puede usarse con el evento "onClick"**.



Captura de Eventos de Usuario: atributo onClick

▶ VENTAJAS USAR onClick:

1. Es sencillo
2. Podríamos **usar este mismo método cuando pulsemos distintos botones, o distintas views**; ya que el método recibe un objeto genérico View, **independiente de si es un button, textview,...**
 - Lo único que tendremos que hacer es después en el método instanciarlo en una variable objeto y hacer el cast correspondiente.

```
public void nombreMetodo (View v){  
    Button botonPulsado = (Button) v;  
    .... ya podemos trabajar con las propiedades del boton  
}
```

▶ INCONVENIENTES USAR onClick:

1. **Sólo sirve para el evento onClick**, no para otros eventos como **onLongClick** (pulsación larga),...

Escuchadores eventos (Event Listener) (I)

- ▶ Esta es la **manera más frecuente de escuchar a los eventos.**
- ▶ Consiste en implementar un **INTERFAZ** que contiene **un método CALLBACK único**, que será llamado cuando se produzca el evento correspondiente.
- ▶ Este **método** va a poder ser **definido desde cualquier clase, siempre que esa clase IMPLEMENTE ESE INTERFAZ.**
- ▶ Ese **listener** debe ser **registrado mediante el método “setOnXXXListener” apropiado.**
- ▶ Existen **muchos EVENT LISTENER.**

EVENT LISTENER (generados por VIEW) (II)

- ▶ Nosotros **veremos los eventos generados por VIEW** (*los hay generados por sensores,...*).
- ▶ La clase **View** contiene una **colección de interfaces anidadas** llamadas "**On<algo>Listener**", cada uno con un **método callback** llamado "**On<algo>()**".
- ▶ Por ejemplo, View.OnClickListener (para manejar "clicks" en un View), View.OnTouchListener (para manejar en un View eventos al tocar la pantalla) y View.OnKeyListener (para manejar dentro de un View los clicks de las teclas en el dispositivo).
- ▶ Por lo tanto **si se quiere que el View sea informado cuando se le haga "click"** (como cuando se selecciona el botón), hay que **implementar el OnClickListener** y **definir el método callback `onClick()`** (donde se realiza la acción cuando se hace el click), y **registrarlo en el View con el `setOnClickListener()`**.

EVENT LISTENER (generados por VIEW) (III)

- ▶ Lo **eventos generados por VIEW** (*los hay generados por sensores,...*) son:

- **onClick()** → *el nombre de los eventos coincide con el de los métodos.*

Para detectar este método desde **cualquier medio**: pantalla táctil, teclas de navegación, trackball,...será necesario implementar el interface **OnClickListener**.

- **onLongClick()**

Para detectar este método será necesario que el usuario pulse **durante más de 1 segundo** y para ello será necesario implementar el interface **OnLongClickListener**.

- **onFocusChange()**

Si el **usuario navega dentro o fuera de un elemento (view)**. Será necesario para detectarlo implementar el interface **OnFocusChangeListener**.

EVENT LISTENER (generados por VIEW) (IV)

- **onKey()**

Cuando **se pulsa o se suelta una tecla**. Es necesario implementar el **interface OnKeyListener**.

- **onTouch()**

Cuando **se pulsa o se suelta o se desplaza en la pantalla táctil**. El evento recibido **contiene información de coordenadas, movimiento,...** Para detectar este evento será necesario implementar el **interface OnTouchListener**.

- **onCreateContextMenu()**

Se llama cuando **se crea un menú de contexto**. Para detectar este evento será necesario implementar el **interface OnCreateContextMenuListener**.

FORMAS DE DEFINIR UN EVENT LISTENER

- ▶ Hay **2 formas principales** de definir o implementar un **Event Listener**:
 1. Crear un objeto anónimo
 2. Implementar en nuestra clase el interfaz con **“implements”**.
- ▶ *En el fondo es lo mismo, pero la programación cambia de una a otra.*

VEREMOS A CONTINUACIÓN LAS FORMAS DE HACERLO Y SUS VENTAJAS O INCONVENIENTES.

FORMA 1: Crear un objeto ANÓNIMO (I)


- ▶ Consiste en **crear un objeto anónimo** que es el **que va a recoger estos eventos**.
- ▶ Vamos a verlo con un ***ejemplo***: supondremos que ***estamos en una activity y tenemos una referencia a un boton llamado "boton"***.

```
public class MiClase extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        // Implementamos el evento "click" del botón
        Button boton = (Button) findViewById (R.id.boton1)
        boton.setOnClickListener ( new OnClickListener() {
            @Override public void onClick(View v) {
                // acciones a realizar cuando se detecta el evento
                ...
            }
        });
    }
    ...
}
```

- Se llama **anónimo** porque el objeto creado (new OnClickListener(...)) no tiene nombre
- **Se crea un objeto por escuchador**, es decir, si tengo 20 botones, o cualquier objeto de tipo view tendría que crear 20 escuchadores (uno para cada objeto)

FORMA 1: Crear un objeto ANÓNIMO (III)

```
public class Ejemplo extends Activity {  
  
    @Override public void onCreate(Bundle valoresGuardados)  
    {  
        Button boton =(Button) findViewById(R.id.boton);  
        boton.setOnClickListener(  
            new OnClickListener() {  
                public void onClick(View view) {  
                    ...  
                }  
            }  
        );  
    }  
}
```



OJO:
MUY
IMPORTANTE

- ❁ **Cuidado**, cuando usemos esta técnica, porque **cuando estamos programando el método “onClick”**, realmente **estamos dentro de una clase diferente a la original (Ejemplo)**.
- ❁ Realmente **estamos en OnClickListener**.
- ❁ Por eso si usamos una referencia como **this**, **no hará referencia al objeto de Ejemplo, sino al de la clase OnClickListener**.

FORMA 2: Implementar el INTERFAZ en nuestra clase (I)

- ▶ Consiste en **implementar en nuestra clase, el INTERFAZ** mediante el uso de “**implements Nombre_Interfaz**”:

```
public class Ej extends Activity implements OnClickListener{  
  
    @Override public void onCreate(Bundle valoresGuardados)  
    {  
        Button boton =(Button) findViewById(R.id.boton);  
        boton.setOnClickListener(this);  
    }  
  
    public void onClick(View v) {  
        ...  
    }  
}
```

this indica que nosotros mismos, esta clase, recoge los eventos.

- ▶ De esta forma, ahora **somos NOSOTROS** los que vamos a recoger los eventos.

FORMA 2: Implementar el INTERFAZ en nuestra clase (II)

► PROBLEMA DE USAR ESTA FORMA:

- Sólo podemos definir un escuchador **onClick**, porque sólo implementamos un **OnClickListener** y, por tanto, sólo podemos tener un método **onClick**.
- *Sin embargo, en el ejemplo anterior podíamos tener un objeto anónimo por cada botón, vista, TextView,... que tengamos.*

► SOLUCIÓN AL PROBLEMA:

- Si hay varios elementos que pueden generar el evento **onClick**, dentro del método **onClick** tendremos que usar un **SWITCH** con varios **CASE** en función del parámetro **View**.
- El parámetro **View** del método **onClick** nos indicará que vista ha generado el evento y así lo podremos gestionar.

FORMA 2: Implementar el INTERFAZ en nuestra clase (III)

▶ VENTAJAS DE ESTA FORMA:

- Consume menos memoria porque no tenemos que generar nuevas clases.
- *Pero es casi inapreciable el consumo que supone la otra forma.*

Manejadores de eventos (Event Handler) (I)

- ▶ Esta forma **se usa si estamos creando un componente personalizado de la clase View** (*extiende de View*).
- ▶ Consiste en **sobreescribir** ciertos **métodos de la clase VIEW**. Hacer @Override sobre el método correspondiente.
- ▶ **Sólo funciona dentro de la clase VIEW** para **reaccionar ante eventos de la clase VIEW**.
- ▶ Es una solución más sencilla que los Event Listener, ya que **no hace falta: ni implementar un Interfaz, ni registrar el método callback**.
- ▶ Nos basta con **implementar en nuestra clase VIEW cualquiera de estos métodos**:

Manejadores de eventos (Event Handler) (II)

onKeyDown(int keyCode, KeyEvent e)

- Llamado cuando una **tecla** es **pulsada**

onKeyUp(int keyCode, KeyEvent e)

- Llamado cuando una **tecla** es **levantada**

onTouchEvent(MotionEvent me)

- Llamado cuando se **utiliza** la **pantalla táctil**

onTrackballEvent(MotionEvent me)

- Llamado **cuando se mueve el trackball**

onFocusChanged(boolean obtengoFoco, int direccion, Rect prevRectanguloFoco)

- Llamado cuando **el View gana o pierde el foco**

Manejadores de eventos (Event Handler) (III)

► Ejemplo de implementación:

```
public class MiVista extends View {  
    ...  
    @Override public boolean onKeyDown(  
        int codigoTecla, KeyEvent evento) {  
        super.onKeyDown(codigoTecla, evento);  
        ...  
        return true; // Hemos procesado el evento  
    }  
  
    @Override public boolean onTouchEvent(MotionEvent evento) {  
        super.onTouchEvent(evento);  
        ...  
        return true; // Hemos procesado el evento  
    }  
}
```

Es **OBLIGATORIO** extender
de la **clase VIEW**

Ver en la **siguiente diapositiva**
EXPLICACIÓN.



Manejadores de eventos (Event Handler) (IV)

► Explicación del código:

Los **métodos**: *onKeyDown, onTouchEvent,...* devuelve un valor **booleano** que será:

- A) **True**: cuando **hayamos terminado de procesar el evento**. *Queríamos hacer algo cuando se pulsase una tecla y ya lo hemos hecho.*
- B) **False**: cuando, *por ejemplo, se pulse una tecla que no nos interesa controlar (ni nos va, ni nos viene)*. Eso hace que **al devolver false el método, el evento se traslade a la vista padre que nos contiene** (otro layout, *por ejemplo*), para que lo gestione ella.

Eventos de Entrada (XIX)
Captura de Eventos de Usuario:
RESUMEN

► Propiedad onClick

- Solo aplicable al evento onClick generado por un view (boton, editText,...) y recogido por una activity donde se desarrollara el método especificado en el atributo onClick de la view

► Evento Listener

- Utilizado por eventos generados por cualquier clase y recogidos por cualquier clase.
- Será necesario implementar el interfaz (onClickListener, onTouchListener,..)
- Registrar un método callback

► Evento Handler

- Aplicables por eventos generados y recogidos por un view
- Será necesario sobescribir el método de la clase view

Eventos de Entrada (XX)

- ▶ Si el dispositivo usa un teclado virtual, normalmente **NO** se lanzará un evento `onKeyDown()`.
- ▶ En general, es una **mala idea crear una interfaz que dependa de teclas específicas.**
- ▶ Para lidiar con estos casos, es **mejor usar acciones** **IME** (como *por ejemplo*, *IME_ACTION_DONE*), para que el método de entrada sepa cómo lidiar con la situación e informe a nuestra aplicación correctamente.

TOUCH MODE

TOUCH MODE (I)

- ▶ Cuando un usuario navega por una **interfaz con teclas direccionales o un trackball**, es necesario que los **items** con los que se pueda interactuar **puedan tomar o perder el foco**.
- ▶ Sin embargo, si el dispositivo tiene **pantalla táctil** y el usuario la empieza a utilizar, **deja de ser necesario darle el foco a las vistas o resaltar elementos**.
- ▶ En cuanto el usuario toca la pantalla, se entra en el denominado **“touch-mode”**.
- ▶ A partir de ahí, **sólo vistas que tengan `isFocusableInTouchMode()` a true serán capaces de ganar foco** (*por ejemplo los campos de texto*).

TOUCH MODE (II)

- ▶ Las otras vistas que pueden tocarse (*botones*, etc.) **no tomarán el foco**, solamente lanzarán los eventos cuando sean apretados.
- ▶ En el momento en el que el **usuario toque una tecla direccional o usa el trackball**, el **dispositivo saldrá del touch mode**, y buscará una vista a la cual darle foco.
- ▶ A partir de ahí, **el usuario podrá interactuar sin tocar la pantalla**.
- ▶ **Este estado es global para el sistema** (*lo comparten todas las ventanas y actividades*).
- ▶ Para consultarlo, llamar a `isInTouchMode()` para saber si el sistema está en touch-mode o no.