

INDICE

1.1. INTRODUCCIÓN	1
1.2. PREPARACIÓN PROYECTO	2
1.3. EMPLEANDO LISTENERS	3
1.3.1. GESTIÓN EVENTOS: EMPLEANDO CLASES ANÓNIMAS	4
1.3.2. GESTIÓN EVENTOS: IMPLEMENTADO LA INTERFACE EN LA ACTIVITY	8
1.3.3. GESTIÓN EVENTOS: IMPLEMENTADO LA INTERFACE EN UNA CLASE SEPARADA	12
1.3.4. GESTIÓN EVENTOS: IMPLEMENTADO LA INTERFACE EN UN OBJETO DENTRO DE LA ACTIVITY	13
1.4. MÚLTIPLES VIEWS PARA EL MISMO EVENTO	14

1.1. Introducción

- ✓ Más información [en este enlace](#).
- ✓ En los móviles, el usuario normalmente interactúa con las aplicaciones tocando sobre la pantalla táctil.

Este tipo de eventos se denominan **Input Events**.

- ✓ La forma en que Android tiene para gestionar los eventos es mediante el uso de Interfaces.
Este tipo de interface se denomina [gestores de eventos](#) o `EventListener`.
- ✓ La idea es que aquel componente que quiera gestionar algún evento, tendrá que registrar la interface asociada para dicho evento.

Una vez registrada (eso se hace haciendo uso de un método, lo veremos a continuación) podremos implementar el código de la Interface, en el que estarán definidos los métodos que conforman el evento a controlar.

Dichos métodos serán llamados por el S.O. Android cuando se produzca el evento sobre el componente gráfico.

- ✓ Para gestionar los eventos de cualquier componente gráfico tendremos que dirigirnos a la página de ese componente y ver cuáles son las interfaces que puede implementar para gestionar los diferentes tipos de evento.

Si visitamos la página de referencia de `AndroidDeveloper` y escogemos la categoría de **Android.View => Interfaces**, podremos ver todos los eventos que pueden gestionar los view y por extensión, cualquiera componente gráfico. Podéis consultar [este enlace](#).

Called when an item is checked or unchecked during selection mode.

Inherited methods

From interface [android.view.ActionMode.Callback](#)

Public methods

onItemCheckedStateChanged added in API level 11

```
public abstract void onItemCheckedStateChanged (ActionMode mode,
        int position,
        long id,
        boolean checked)
```

Called when an item is checked or unchecked during selection mode.

Parameters

mode	ActionMode ; The ActionMode providing the selection mode
position	int ; Adapter position of the item that was checked or unchecked
id	long ; Adapter ID of the item that was checked or unchecked
checked	boolean ; true if the item is now checked, false if the item is now unchecked.

- ✓ Como dijimos antes, los eventos en que el usuario interacciona son los Event Listener.

Estos eventos se encuentran definidos en interfaces y dentro de ellas están definidos algunos de los siguientes métodos:

- ✓ Debemos indicar que pueden producirse varios eventos 'a la vez' y estos son gestionados por medio de una cola FIFO.

Por ejemplo, cuando pulsamos una lista en la pantalla y mantenemos pulsado durante un rato, normalmente aparece un menú emergente con opciones sobre la lista.

En este se producen dos eventos: OnLongClick y Click sobre la lista. Primero se lanzará el 'Long Click' y después el 'Click'.

1.2. Preparación proyecto

- ✓ Comenzamos creando un nuevo proyecto: **Eventos**
- ✓ Para informar al usuario vamos a hacer uso de la [clase Toast](#).

Es un tipo de notificación y sirve para mostrar un mensaje corta que aparece y desaparece automáticamente. No requiere intervención por parte do usuario.



- ✓ La forma más sencilla de llamarla es:

```
Toast.makeText(getApplicationContext(), "AVISO", Toast.LENGTH_LONG).show();
```

- ✓ El primer parámetro es una referencia al contexto. El contexto es una interface (todas las activities derivan de ella) que nos va a servir para acceder a los recursos da aplicación.

Normalmente se obtiene llamando al [método getApplicationContext\(\)](#)

- ✓ El segundo parámetro es el texto que queremos mostrar al usuario.

Recordar que vuestro proyecto tenéis que tener soporte para varios idiomas, por lo tanto los mensajes tienen que referenciarse da forma: **R.string**.

Si queremos una variable local o el valor de alguna de las cadenas de texto creadas en /res/values deberemos emplear el método getResources () de la forma: String mensaje = getResources ().getString (R.string.app_name);

- ✓ El tercer parámetro es el tempo que se va a mostrar el mensaje. Puede ser Toast.LENGTH_LONG o Toast.LENGTH_SHORT
- ✓ Por último se llama al método show() del objeto creado para mostrar el mensaje.

Nota: En los ejemplos siguientes se va a hacer uso de la interface que gestiona los eventos de click sobre los botones, pero los conceptos y cómo hacerlo se puede aplicar a cualquier tipo de evento.

1.3. Empleando Listeners

- ✓ Un **Event Listener** es una interface de la clase Vista (View) que contiene un único método de tipo callback que hay que implementar.
- ✓ Android invoca estos métodos cuando la Vista detecta que el usuario está provocando un tipo concreto de interacción con ese elemento de la interface de usuario.
- ✓ Existen los siguientes métodos callback:

- **onClick ()**: de **View.OnClickListener**. Este método se invoca cuando el usuario toca un elemento con un dedo (modo contacto), hace click con la bola de navegación (TrackBall) del dispositivo o pulsa la tecla "Enter" estando en un componente.
 - **onLongClick()**: de **View.OnLongClickListener**. Este método se llama cuando el usuario toca y mantiene el dedo sobre un elemento (modo de contacto), hace click sin soltar con la bola de navegación (TrackBall) o pulsa la tecla "Enter" durante un segundo estando en un elemento.
 - **onFocusChange()**: de **View.OnFocusChangeListener**. Se invoca cuando el usuario mueve el cursor cara una Vista o se aleja de esta utilizando la bola de navegación o usando las teclas de navegación.
 - **onKey()**: de **View.OnKeyListener**. Se llama cuando el usuario se centra en un elemento y se pulsa o libera una tecla del dispositivo.
 -
- ✓ Estos métodos son los únicos que se van implementar en sus respectivas interfaces.
 - ✓ Estas interfaces tienen el formato: **on...Listener()**
 - ✓ Una vez que tenemos implementada la interface la tenemos que pasar como parámetro a la vista (view) correspondiente a través de **vista.set...Listener()**.
 - ✓ Recordar que en java los interfaces son clases abstractas que solo definen los atributos y métodos (con los parámetros) que va a tener esa clase pero no los implementa. Por tanto siempre que se implemente una interface hay que implementar los métodos que define.
 - ✓ En este caso la interface View.OnClickListener está definida como sigue:

<http://developer.android.com/reference/android/view/View.OnClickListener.html>

```
1 public static interface View.OnClickListener {
2 public abstract void onClick (View v);
3 }
```

- ✓ En este ejemplo, cuando implementemos la interface **View.OnClickListener** debemos implementar el método **onClick**.
- ✓ A continuación, a modo de ejemplo, vamos a ver 3 formas de implementar esta interface. Lo que se haga con esta se puede hacer de forma semejante con cualquier otra.

1.3.1.Gestión Eventos: Empleando Clases anónimas

- ✓ Dentro del paquete **Eventos** crear una nueva 'Empty Activity' de nombre: **UD03_01_Eventos** de tipo Launcher.

Modifica el archivo **AndroidManifest.xml** y añade una label a la activity.

Recordar que si estáis empleando las bibliotecas de compatibilidad (por defecto), **vuestra activity derivará de AppCompatActivity** y no de Activity

- ✓ Como dijimos en la introducción, tenemos que registrar una interface con un view, enviando como dato un objeto que implemente dicha interface.

Podemos pasarle como parámetro al escuchador el código que implementa el objeto. De esta forma, no se crea ni una clase ni un objeto de modo explícito a que se le asocie un nombre. Sino que se crea un objeto de tipo OnClickListener al que no se le asocia ningún nombre, de ahí lo de **anónimo**.

- ✓ Para gestionar cualquiera de los eventos tenemos que asociar dicho evento al view, por medio de un registro de una de las interfaces anteriores, donde estará definido el método al que el S.O. llamará cuando se produzca el evento.

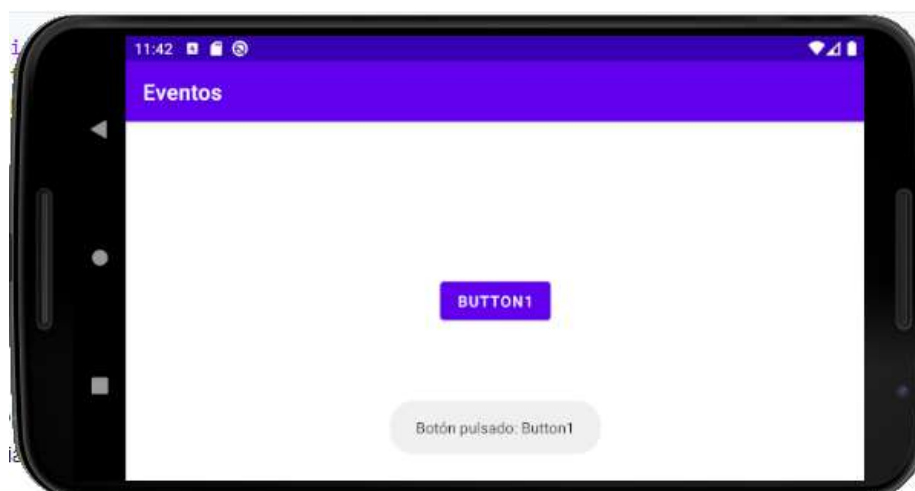
Para eso podemos hacer uso de las **Clases anónimas**.

La forma de implementar de forma anónima una interface es: `objetoView.setXXXXXXListener(new XXXXXXXXListener(){});`

- ✓ Como vemos, tenemos:
 - Tener una referencia a la view sobre el que queremos gestionar el evento.
 - Llamar al método `setXXXXInterface`, por ejemplo, `setOnClickListener` sería el evento para gestionar el evento de click sobre un botón.
 - Dicho método espera como parámetro un objeto de una clase que implemente la interface. En nuestro caso haremos uso de las clases anónimas y haremos un **new XXXXXXXXListener**, es decir, de la interface. En el caso anterior del botón, la interface se llama OnClickListener (mirar imagen anterior).
 - Al hacer el new, automáticamente el Android Studio implementa los métodos de la Interface. En el caso del ejemplo, el método `onClick`.

Veamos un ejemplo.

- ✓ Vamos a hacer una activity con un botón y vamos gestionar su evento de 'click' mostrando un mensaje (notificación) de tipo Toast.



Código XML del layout de la activity:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".UD03_01_Eventos">
    <Button
        android:id="@+id/btnAceptar_UD03_01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:text="Button"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Tenemos un botón de nombre **btnAceptar_UD03_01**.

- ✓ Ahora en el código de la Activity haremos los pasos indicados para gestionar el evento Click():

Siempre **después de la llamada a setContentView del método onCreate** (que es donde asociamos el layout a la activity):

```
package com.example.eventos;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.Button;

public class UD03_01_Eventos extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_u_d03_01_eventos);

        // Obtenemos la referencia al botón por medio del método findViewById
        Button btnAceptar = findViewById(R.id.btnAceptar_UD03_01);
    }
}
```

Hacemos uso del [método findViewById](#) para obtener una referencia al botón.

Como parámetro lleva un número. Este lo obtenemos de la clase R la cual cada vez que compilamos la aplicación, asocia cada recurso de la carpeta /res con un número.

Por eso es importante que el nombre del control gráfico sea identificativo, ya que podemos hacer referencia a controles de otras activities y por lo tanto tendríamos un valor null para el control.

- ✓ Ahora llamamos al [método setOnClickListener de la clase View](#) tendremos que pasarle como parámetro un objeto de una clase que implemente la [interface OnClickListener](#) o bien crear de forma anónima un objeto de dicha interface, que es lo que haremos:

Nota: Al hacer 'enter' en Android Studio cuando sea el nombre de la interface, ya genera automáticamente los métodos de la misma.

```
package com.example.eventos;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class UD03_01_Eventos extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_u_d03_01_eventos);

        // Obtenemos la referencia al botón por medio del método findViewById
        Button btnAceptar = findViewById(R.id.btnAceptar_UD03_01);

        btnAceptar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

            }
        });
    }
}
```

- ✓ Ahora dentro del método onClick podremos poner el código que queramos.

Comentar que dicho método lleva como parámetro un objeto de la clase View que es el objeto que tiene registrado el evento y sobre el cual se hizo el evento (en este caso el botón).

Podemos hacer un **cast** de dicho objeto para acceder a todas las propiedades y métodos del botón, aunque en el caso de las interfaces anónimas no tiene mucho sentido ya que, como en el ejemplo, podemos acceder a btnAceptar que es el botón.

```
btnAceptar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Button btn = (Button)v; // Sería el mismo que btnAceptar
        Toast.makeText(getApplicationContext(), "Botón pulsado: "+
```

```
btn.getText(), Toast.LENGTH_LONG).show();

    }
});
```

Aviso: Si elegís esta forma de gestionar los eventos, deberéis de hacerlo en un método aparte donde estén las llamadas a `setOnClickListener`, y llamarlo desde el método `onCreate`.

1.3.2.Gestión Eventos: Implementado la interface en la Activity

- ✓ La segunda opción sería la de crear una clase que implemente dicha interface (en el ejemplo, `ActionListener`) y pasarle un objeto de dicha clase al método `setOnClickListener`.

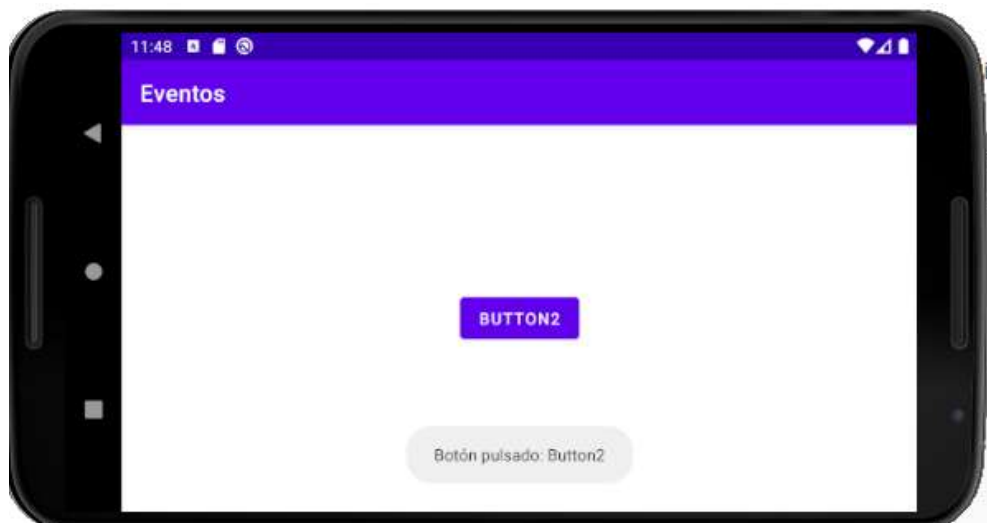
De forma más rápida podemos implementar dicha interface en la propia activity.

Veamos un ejemplo.

- ✓ Dentro del paquete **Eventos** crear una nueva 'Empty Activity' de nombre: **UD03_02_Eventos** de tipo Launcher.

Modifica el archivo **AndroidManifest.xml** y añade una label a la activity.

- ✓ Vamos a hacer una activity con un botón y vamos a gestionar su evento de 'click' mostrando un mensaje (notificación) de tipo Toast, como hicimos antes, pero implementando el código de otra forma.



Código XML del layout de la activity:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">
```



```

    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".UD03_02_Eventos">

    <Button
        android:id="@+id/btnAceptar_UD03_02"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:text="Button"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>

```

✓ Ahora en el código de la clase implementamos la interface (cuidado con la Interface ClickListener.

Al empezar a escribir el nombre de la interface el asistente mostrará diversas opciones.

```

package com.example.eventos;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

public class UD03_02_Eventos extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_u_d03_02__eventos);
    }
}

```

Debemos escoger la que deriva de la clase View

```

package com.example.eventos;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;

public class UD03_02_Eventos extends AppCompatActivity implements View.OnClickListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_u_d03_02__eventos);
    }
}

```

Os dará un error:

Implementando una interface

```
package com.example.eventos;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;

public class UD03_02_Eventos extends AppCompatActivity implements View.OnClickListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_u_d03_02_eventos);
    }
}
```

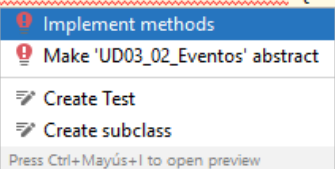
El error aparece ya que debemos de implementar los métodos de la interface, pero eso lo hace Android Studio de forma automática. Debemos situarnos sobre la línea y pulsar las teclas **alt+Enter**.

```
package com.example.eventos;

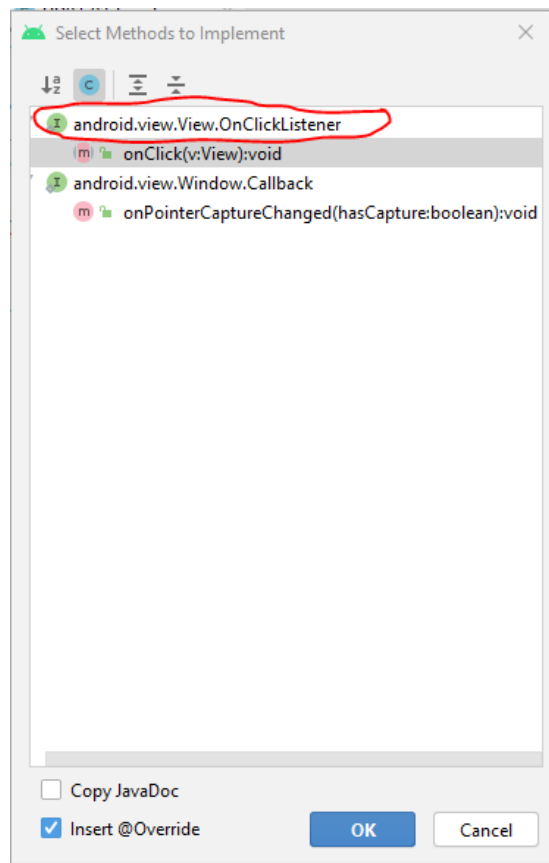
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;

public class UD03_02_Eventos extends AppCompatActivity implements View.OnClickListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_u_d03_02_eventos);
    }
}
```



Escogemos la opción **Implement Methods**



Escogeremos el de la interface correspondiente. En este caso OnClickListener de la clase View.

✓ El código de la clase queda como sigue:

```
package com.example.eventos;
```

```
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
```

```
public class UD03_02_Eventos extends AppCompatActivity implements View.OnClickListener {
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_u_d03_02_eventos);
    }
```

```
    @Override
    public void onClick(View v) {
        Toast.makeText(getApplicationContext(), "Botón pulsado: " + ((But-
ton)v).getText(), Toast.LENGTH_LONG).show();
    }
}
```

Fijarse como ahora estoy haciendo un cast sobre el objeto y sin la necesidad de tener una variable intermedia como hicimos antes.

- ✓ Ahora queda registrar esta clase como la que tiene que gestionar los eventos del botón:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_u_d03_02_eventos);

    Button btnAceptar = findViewById(R.id.btnAceptar_UD03_02); // Recordar poner el botón 2.
    Si ponéis el 1 btnAceptar vale null
    btnAceptar.setOnClickListener(this);
}

@Override
public void onClick(View v) {
    Toast.makeText(getApplicationContext(), "Botón pulsado: " + ((Button)v).getText(), Toast.LENGTH_LONG).show();
}
```

- ✓ Nota: Se empleamos varios botones y registramos sus eventos de click, todos irán al mismo método.

Para diferenciar un de otro haremos uso del id asociado al view, de la forma v.getId() (siendo v el objeto de la clase View que está definido como parámetro en el método onClick).

1.3.3.Gestión Eventos: Implementado la interface en una clase separada

- ✓ En este caso crearíamos una clase que implemente la interface da forma:

```
public class GestionEventos implements View.OnClickListener {
    @Override
    public void onClick(View v) {
        Button btn = (Button) v;
        Toast.makeText(v.getContext(), "Pulsaste " + btn.getText(), Toast.LENGTH_SHORT).show();
    }
}
```

- **Línea 8:** Fijarse que dentro de esta clase el Context no existe, ya que no estamos en una Activity. Para obtener el Context tenemos que valernos del View que viene como parámetro y que representa el View que provocó el evento.

- ✓ Ahora en la Activity, registraríamos los eventos de los views de la siguiente forma:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_u_d03_03_eventos);
    GestionEventos gestionEventos = new GestionEventos();
    Button boton3 = (Button) findViewById(R.id.btnAceptar_UD03_03);
    boton3.setOnClickListener(gestionEventos);

    Button boton4 = (Button) findViewById(R.id.btnAceptar_UD03_04);
    boton4.setOnClickListener(gestionEventos);
}
```

- ✓ Observar la **Línea 1** como se crea un objeto de la clase GestionEventos que implementa la interface y después se asocia cada botón a dicho objeto.

Esta forma es un *engorro*, pues no tenemos acceso a la Activity (no directamente) por lo que no podemos emplear las propiedades definidas en ella.

1.3.4.Gestión Eventos: Implementado la interface en un objeto dentro de la Activity

- ✓ Las interfaces no dejan de ser clases, por lo que podemos crear un objeto de dicha clase dentro de la Activity de la forma:
- ✓ `_OnClickListener` es un objeto de tipo `OnClickListener`.

```
private View.OnClickListener _OnClickListener = new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        Button btn = (Button) v;
        Toast.makeText(v.getContext(), "Pulsaste: " + btn.getText(), Toast.LENGTH_SHORT).show();
    }
};
```

- ✓ Pasamos ese objeto a las vistas que deseemos cuando se haga Click en ellas.

```
Button boton5 = (Button) findViewById(R.id.btnAceptar_UD03_05);
boton5.setOnClickListener(_OnClickListener);

Button boton6 = (Button) findViewById(R.id.btnAceptar_UD03_06);
boton6.setOnClickListener(_OnClickListener);
```

1.4. Múltiples views para el mismo evento

- ✓ Es la forma más sencilla de desencadenar una acción.

No es muy aconsejable ya que estamos mezclando la vista (es decir, el diseño del layout) con la forma de controlar los eventos.

- ✓ Consiste en poner como propiedad del view en el layout lo siguiente: **android:onClick="nombre_método"**

El nombre del método es el nombre que tendrá que definirse en el código de la activity.

Solamente hay que poner el nombre, sin parámetros de ningún tipo.

- ✓ En el código de la activity hay que definir un método:
 1. Con modificador de acceso 'public'
 2. Que no devuelva nada: 'void'
 3. Con el mismo nombre que se está usando en el layout.
 4. Con un parámetro de tipo view, que será el view que provoco un evento

```
public void nombreMétodo (View v) {  
2  
3 }
```

- ✓ Indicar que esta forma [se puede aplicar a cualquier View](#) en la que queramos controlar el evento de click sobre ella.
Veamos un ejemplo.
- ✓ Dentro del paquete **Eventos** crear una nueva 'Empty Activity' de nombre: **UD03_04_Eventos** de tipo Launcher.
Modifica el archivo **AndroidManifest.xml** y añadir la una label a la activity.

Gestión de eventos en el layout



La aplicación tiene dos botones y un texto.



Al pulsar el botón 1, el color del texto pasa a amarillo.



Al pulsar el botón 2, el color del texto pasa a azul.

✓ Código XML del layout:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".UD03_03_a_Eventos">
```

```
<Button
    android:id="@+id/btnAcep1_UD03_04_Event"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:text="Aceptar 1"
    android:onClick="pulsarBoton"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

```
<Button
    android:id="@+id/btnAcep2_UD03_04_Event"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginEnd="8dp"
    android:layout_marginTop="8dp"
```



```

        android:text="Aceptar 2"
        android:onClick="pulsarBoton"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/txtResultado_UD03_04_Event"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:text="Es un texto de prueba"
    android:textSize="36sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>

```

✓ El código de la activity:

```

package com.example.eventos;

import androidx.appcompat.app.AppCompatActivity;
import android.graphics.Color;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class UD03_03_a_Eventos extends AppCompatActivity {

    public void pulsarBoton(View v) {
        final TextView texto = findViewById(R.id.txtResultado_UD03_04_Event);
        switch (v.getId()) {
            case R.id.btnAcep1_UD03_04_Event:
                texto.setTextColor(Color.YELLOW);
                break;
            case R.id.btnAcep2_UD03_04_Event:
                texto.setTextColor(Color.BLUE);
        }
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_u_d03_04__eventos);
    }
}

```