

EVENTOS

1. INTRODUCCION

- Hasta ahora hemos definido interfaces de usuario para poder crear pantallas en las aplicaciones, pero no podíamos interaccionar con ellas porque no hemos definido código para **responder a eventos**.
- De eso trataremos en este documento: veremos cómo responder ante distintas acciones que el usuario aplique sobre la pantalla y a leer valores que se introduzcan en las vistas.
- Las aplicaciones Android suelen estar **controladas por eventos**. A diferencia de otro tipo de aplicaciones, las aplicaciones controladas por eventos se inician y quedan a la espera de registrar dichos eventos, por ejemplo, registrar cuando se produce la pulsación de un botón por parte del usuario. Además, tanto el propio sistema operativo como una aplicación pueden iniciar eventos, pero los más significativos son los que inicia el propio usuario.

2. REPUESTA A LA PULSACION DE UN BOTON

- Cada vez que hacemos clic en uno de los botones de un layout, se produce un **evento onClick**, que llama al método con su mismo nombre.
- Los métodos que reciben eventos onClick deben ser de tipo público y nulos (*public void*). Y además deben aceptar como parámetro un objeto de tipo View, que referencia al elemento de la interfaz sobre el que hemos hecho clic.
- Existen diferentes formas de gestionar los eventos de un botón.
Para probarlas, recuperaremos el proyecto **Navegador** (del cual tenemos tres versiones), y lo modificaremos para que añada un mensaje con información sobre qué botón se ha pulsado.



La primera modificación será añadir una TextView en el archivo de layout para visualizar el mensaje de respuesta a cada pulsación.

```
<TextView
    android:id="@+id/respuesta"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="" />
```

2.1. Gestionar el evento onClick mediante la propiedad onClick del elemento Button en el fichero .xml

- Es la opción más simple.
- Añadimos la propiedad **android:onClick** a cada botón:

```
<Button
    android:id="@+id/siBtn"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Sí"
    android:onClick="onClickSi"/>
<Button
    android:id="@+id/noBtn"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="No"
    android:onClick="onClickNo"/>
<Button
    android:id="@+id/avecesBtn"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="A veces"
    android:onClick="onClickAVeces"/>
```

- El nombre indicado como valor del atributo "*android:onClick*" debe corresponderse con un método público y nulo. Así, en este ejemplo, debemos declarar los tres métodos públicos:

```
public void onClickSi(View v)
public void onClickNo(View v)
public void onClickAVeces(View v)
```

- Para trabajar con los elementos de pantalla desde el código, lo primero que hay que hacer es **recuperar el objeto** que los representa, mediante el método ***findViewById(identificador)***. Este método busca en el layout mostrado el elemento que tenga como identificador el que se haya pasado como parámetro. Y devuelve un objeto de tipo View, por lo que es necesario hacer una conversión explícita o cast a la clase correspondiente del objeto, para poder acceder a los métodos que sean necesarios:

```
TextView respuesta = (TextView) findViewById(R.id.respuesta);
```

- Y, por último, en cada método encargado de gestionar la pulsación de cada botón, se escribe el mensaje en el TextView de salida mediante el método ***setText()***.

```
respuesta.setText("Has hecho click en SI");
```

- Código resultante

```
package com.example.user.navegador_linearlayout;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends Activity {

    private TextView lblRespuesta;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        /* usamos findViewById, que recibe como parámetro una constante de la clase R
        y devuelve un objeto de tipo View, el cual debe ser convertido mediante
        casting a un objeto de tipo TextView */
        lblRespuesta = (TextView) findViewById(R.id.respuesta);
    } //end onCreate

    // creamos un método para cada boton
    public void onClickSi (View v) {
        lblRespuesta.setText("Has hecho click en SI");
    } //end pulsarBtnSi

    ... y así con los demás ...
}
```

Modificación en Navegador_RelativeLayout.

- En lugar de crear un método para cada botón, se puede crear un único método, común para todos los botones.

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

- Y, por código, controlar qué botón fue el que se pulsó, mediante el método **getId()**.

```
package com.example.user.navegador_relativelayout;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends Activity {

    private TextView lblRespuesta;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        /* usamos findViewById, que recibe como parámetro una constante de la clase R
        y devuelve un objeto de tipo View, el cual debe ser convertido mediante casting a
        un objeto de tipo TextView */
        lblRespuesta = (TextView) findViewById(R.id.respuesta);
    }
}
```

```
// creamos un método para todos los botones --> controlar qué botón fue el que se pulsó
public void onClickBtn(View v) {
    // usamos el metodo getId(), que devuelve el identificador de la View pasada
    // como parámetro
    if (v.getId() == R.id.siBtn)
        lblRespuesta.setText("Has hecho click en SI");
    else if (v.getId() == R.id.noBtn)
        lblRespuesta.setText("Has hecho click en NO");
    else if (v.getId() == R.id.avecesBtn)
        lblRespuesta.setText("Has hecho click en A VECES");

    /* o con switch
    switch (v.getId()) {
        case R.id.siBtn:
            lblRespuesta.setText("Has hecho click en SI");
            break;
        case R.id.noBtn:
            lblRespuesta.setText("Has hecho click en NO");
            break;
        case R.id.avecesBtn:
            lblRespuesta.setText("Has hecho click en A VECES");
            break;
    } // end switch
    */
} // end onClickBtn
} // end Activity
```

Comentario

Con este código es suficiente para que cada botón imprima su propio mensaje. Ahora bien, aunque es una opción sencilla no es la mejor opción.

Si definimos los eventos en el layout pero no los métodos en la Activity, el código compilará sin ningún problema y únicamente fallará en tiempo de ejecución cuando se percate de que el/los métodos no existen.

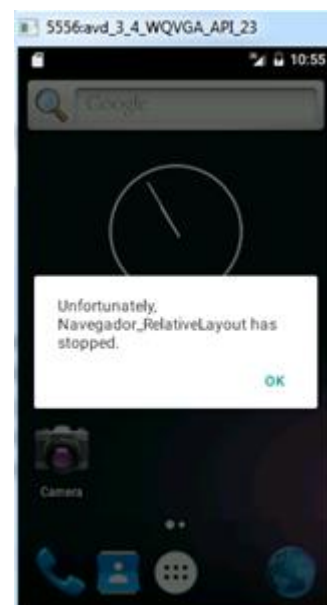
Por ejemplo, si sustituimos

```
public void onClickBtn(View v) {...}
```

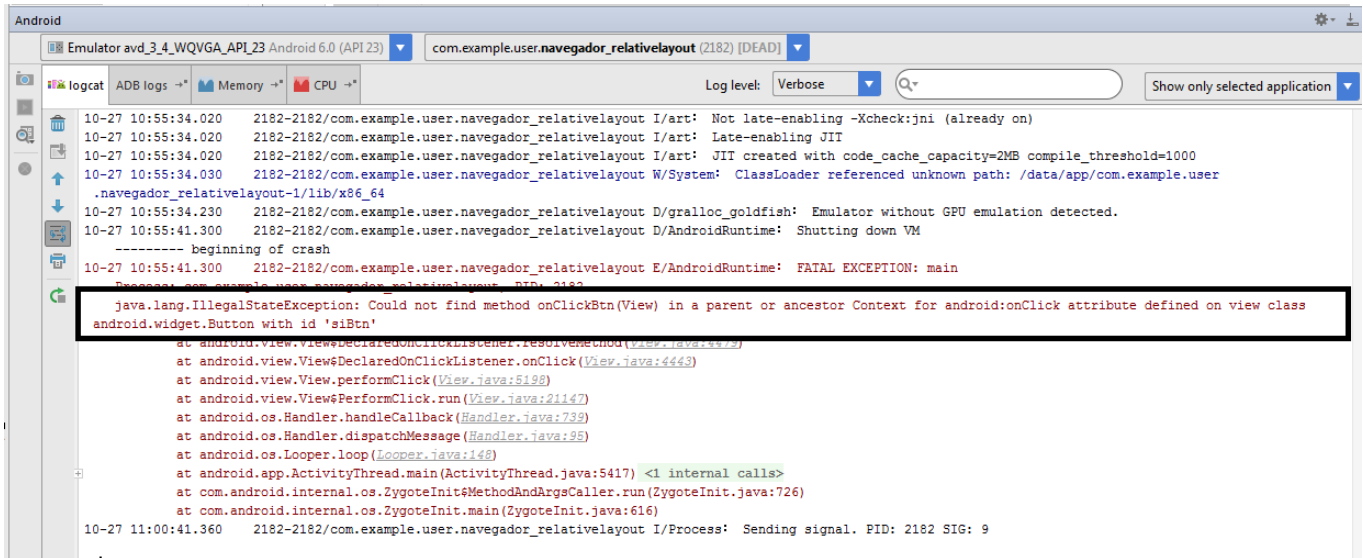
por

```
public void onClickbtn (View v) {...}
```

La aplicación es lanzada sin problema, pero aborta cuando pulsamos uno de los botones:



Consultamos el archivo de log para ver el error que se ha producido:



```
10-27 10:55:34.020 2182-2182/com.example.user.navegador_relativelayout I/art: Not late-enabling -Xcheck:jni (already on)
10-27 10:55:34.020 2182-2182/com.example.user.navegador_relativelayout I/art: Late-enabling JIT
10-27 10:55:34.020 2182-2182/com.example.user.navegador_relativelayout I/art: JIT created with code_cache_capacity=2MB compile_threshold=1000
10-27 10:55:34.030 2182-2182/com.example.user.navegador_relativelayout W/System: ClassLoader referenced unknown path: /data/app/com.example.user.navegador_relativelayout-1/lib/x86_64
10-27 10:55:34.230 2182-2182/com.example.user.navegador_relativelayout D/gralloc_goldfish: Emulator without GPU emulation detected.
10-27 10:55:41.300 2182-2182/com.example.user.navegador_relativelayout D/AndroidRuntime: Shutting down VM
----- beginning of crash -----
10-27 10:55:41.300 2182-2182/com.example.user.navegador_relativelayout E/AndroidRuntime: FATAL EXCEPTION: main
Process: com.example.user.navegador_relativelayout, PID: 2182
java.lang.IllegalStateException: Could not find method onClickBtn(View) in a parent or ancestor Context for android:onClick attribute defined on view class android.widget.Button with id 'siBtn'
    at android.view.View$DeclaredOnClickListener.resolveMethod(View.java:4479)
    at android.view.View$DeclaredOnClickListener.onClick(View.java:4443)
    at android.view.View.performClick(View.java:5198)
    at android.view.View$PerformClick.run(View.java:21147)
    at android.os.Handler.handleCallback(Handler.java:739)
    at android.os.Handler.dispatchMessage(Handler.java:95)
    at android.os.Looper.loop(Looper.java:148)
    at android.app.ActivityThread.main(ActivityThread.java:5417) <1 internal calls>
    at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:726)
    at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:616)
10-27 11:00:41.360 2182-2182/com.example.user.navegador_relativelayout I/Process: Sending signal. PID: 2182 SIG: 9
```

2.2. Añadir un listener con una clase anónima

Págs 49-50 del Manual de SG Oliver

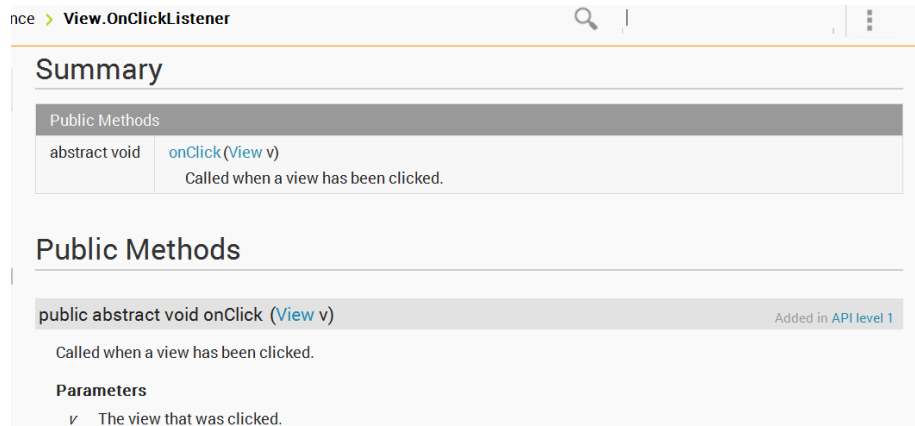
Págs 163-165 del Manual de JArroba

- Cuando una aplicación se encuentra a la espera de que se produzca un determinado evento, se dice que está “a la escucha” de dicho evento. Así, el objeto que responde a un evento se denomina “escuchador” o “listener”.
- Un **listener** (escuchador de eventos) es una interfaz de la clase **View** que contiene un método que hay que sobrescribir.
- El SDK de Android incorpora interfaces de escuchador para diferentes eventos. Esto da lugar a que se puedan capturar diferentes eventos mediante listeners. P.ej. los eventos **onClick** (el que nos interesa en este momento referido a la pulsación de un botón), **onCreateContextMenu** (para crear un menú contextual), **onItemSelected** (para controlar cuando ha sido pulsado un elemento de una lista o spinner)...
- Interfaces de la clase View:

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

View.OnClickListener

Interface definition for a callback to be invoked when a view is clicked.



- Ejemplo de código:

```
Button btnSi = (Button) findViewById(R.id.siBtn);
btnSi.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        respuesta.setText("Has hecho click en SI");
    }
});
```

Comentario:

- La primera línea crea un objeto (btnSi) que apunta al botón cuyo evento queremos gestionar.
- En las siguientes:
 - Llamamos al método Listener (“escuchador”) del objeto btnSi: **setOnClickListener()**. Este método será llamado cuando el usuario haga click sobre el botón.
 - Se le pasa como parámetro la creación de una clase anónima que implementa una interfaz (**OnClickListener**), para la cual hay que sobrescribir el único método que tiene dicha interfaz: **onClick()**.

Modificación en Navegador_TableLayout.

- Para esta versión, eliminamos la propiedad **android:onClick** del layout.

```
/* VERSION QUE GESTIONA EL EVENTO DEL BOTON MEDIANTE CLASES ANONIMAS */
/* ===== */
package com.example.user.navegador_tablelayout;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends Activity {
```

```

private TextView lblRespuesta;
private Button btnSi, btnNo, btnAveces;

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

    //Obtenemos una referencia a los controles de la interfaz
    lblRespuesta = (TextView) findViewById(R.id.respuesta);
    btnSi = (Button) findViewById(R.id.siBtn);
    btnNo = (Button) findViewById(R.id.noBtn);
    btnAveces = (Button) findViewById(R.id.avecesBtn);

    // Gestión del evento de cada boton mediante clases anonimas:
    btnSi.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            lblRespuesta.setText("Has hecho click en SI");
        }
    });
    ... y así con los demás ...

} //end onCreate
} //end activity

```

- Una forma equivalente de atender al evento sería crear un objeto que implemente la interfaz de la clase correspondiente; en este caso, la interfaz es el escuchador **OnClickListener**. Y enlazarlo con la View mediante el método **setOn____Listener()** de dicha View: en este caso, la View es nuestro botón y el método es **setOnClickListener()**.

Código con esta nueva opción

```

/* VERSION QUE GESTIONA EL EVENTO DEL BOTON MEDIANTE CLASES ANONIMAS 2ª forma*/
/* ===== */
package com.example.luli.navegador_v4;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends Activity {
    private TextView lblRespuesta;
    private Button btnSi, btnNo, btnAveces;

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

        //Obtenemos una referencia a los controles de la interfaz
        lblRespuesta = (TextView) findViewById(R.id.respuesta);
        btnSi = (Button) findViewById(R.id.siBtn);
        btnNo = (Button) findViewById(R.id.noBtn);
        btnAveces = (Button) findViewById(R.id.avecesBtn);

        // Asociamos cada escuchador a su botón
        btnSi.setOnClickListener(objetoEscuchador);
    }
}

```

```

        btnNo.setOnClickListener(objetoEscuchador2);
        ...
    } //end onCreate

    // Creamos objeto de tipo escuchador
    private View.OnClickListener objetoEscuchador = new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            lblRespuesta.setText("Has hecho click en SI");
        }
    };

    private View.OnClickListener objetoEscuchador2 = new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            lblRespuesta.setText("Has hecho click en NO");
        }
    };
    ...
} //end Activity

```

- De esta forma, también se puede crear un **único escuchador** y asignarlo a los diferentes botones.

Código con esta nueva opción

```

/* VERSION QUE GESTIONA EL EVENTO DEL BOTON MEDIANTE CLASES ANONIMAS y UNICO LISTENER*/
/* ===== */
package com.example.user.navegador_v5;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends Activity {
    private TextView lblRespuesta;
    private Button btnSi, btnNo, btnAveces;

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

        //Obtenemos una referencia a los controles de la interfaz
        lblRespuesta = (TextView) findViewById(R.id.respuesta);
        btnSi = (Button) findViewById(R.id.siBtn);
        btnNo = (Button) findViewById(R.id.noBtn);
        btnAveces = (Button) findViewById(R.id.avecesBtn);

        // Asociamos cada escuchador a su botón
        btnSi.setOnClickListener(objetoEscuchador);
        btnNo.setOnClickListener(objetoEscuchador);
        ...
    } //end onCreate

    // Creamos objeto escuchador para los tres botones
    private View.OnClickListener objetoEscuchador = new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if (v.getId()==R.id.siBtn)
                lblRespuesta.setText("Has hecho click en SI");

```



```

        else if (v.getId()==R.id.noBtn)
            lblRespuesta.setText("Has hecho click en NO");
        else if (v.getId()==R.id.avecesBtn)
            lblRespuesta.setText("Has hecho click en A VECES");
    }
};
} //end Activity

```

3. RESPUESTA A LA PULSACION DE OTROS TIPOS DE BOTON

3.1. ToggleButton

- Como son botones, también disponen de la propiedad **android:onClick**.
- Como tienen dos estados, suele ser de utilidad conocer en qué estado ha quedado el botón tras ser pulsado, para lo que podemos utilizar su método **isChecked()**.

Para probarlo, recuperaremos el proyecto **Componentes**.

- **Ejemplo de código:**

Archivo de layout

```

<ToggleButton
    android:id="@+id/tb_1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOff="Off"
    android:textOn="On"
    android:onClick="onClickToggle"/>

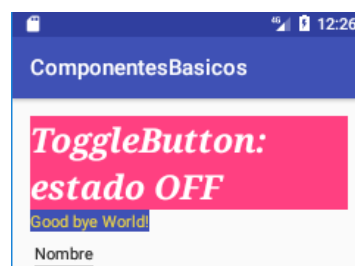
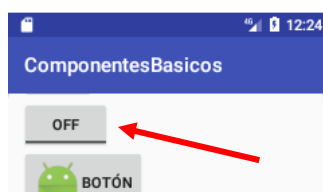
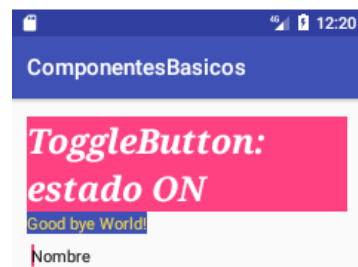
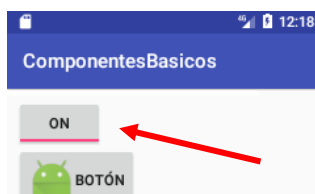
```

Actividad principal

```

public void onClickToggle(View v){
    if(tb_1.isChecked())
        tv_1.setText("ToggleButton: estado ON");
    else
        tv_1.setText("ToggleButton: estado OFF");
}

```



También se puede gestionar el evento

mediante

un **listener con clase anónima**, y eliminando la propiedad **android:onClick** del archivo .xml:

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

    mensaje = (TextView) findViewById (R.id.LblMensaje);
    btnToggle = (ToggleButton) findViewById (R.id.btnToggle);

    btnToggle.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if(btnToggle.isChecked())
                mensaje.setText("Click en toggleButton. Su estado es ON");
            else
                mensaje.setText("Click en toggleButton. Su estado es OFF");
        }
    });
} //end onCreate
```

3.2. ImageButton

- Funciona igual que un botón normal.
- También funcionaría lo mismo si el botón se compone de texto más imagen.
- **Ejemplo de código:**

```
//listener del ImageButton
imgBtn_1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        tv_1.setText("ImageButton pulsada");
    }
});
```

3.3. CheckBox

- Como ya se comentó en un documento anterior, este componente es similar al componente ToggleButton.
- Se puede comprobar su estado mediante el método **isChecked()**.
- Se puede establecer un estado concreto mediante el método **setChecked(estado)**.
- **Evento onClick:**
 - Para determinar el estado de una casilla de verificación podemos gestionar su evento **onClick**. Si el método **isChecked()** de la casilla devuelve *true*, significa que está activada. En caso contrario, devolverá el valor *false*.
 - Ejemplo (con la primera casilla de verificación):

```
<CheckBox
    android:id="@+id/chk_1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Windows"
    android:onClick="onClickChk1"/>
```

```
//Listener del primer checkbox
public void onClickChk1 (View v){
    if ( ((CheckBox)v).isChecked() )
        tv_1.setText("Chequeado: Windows");
    else
        tv_1.setText("Desmarcado Windows");
}
```

- **Evento onCheckedChanged :**

- Informa de que ha cambiado el estado de la casilla de verificación.
- Ejemplo (con la segunda casilla de verificación):

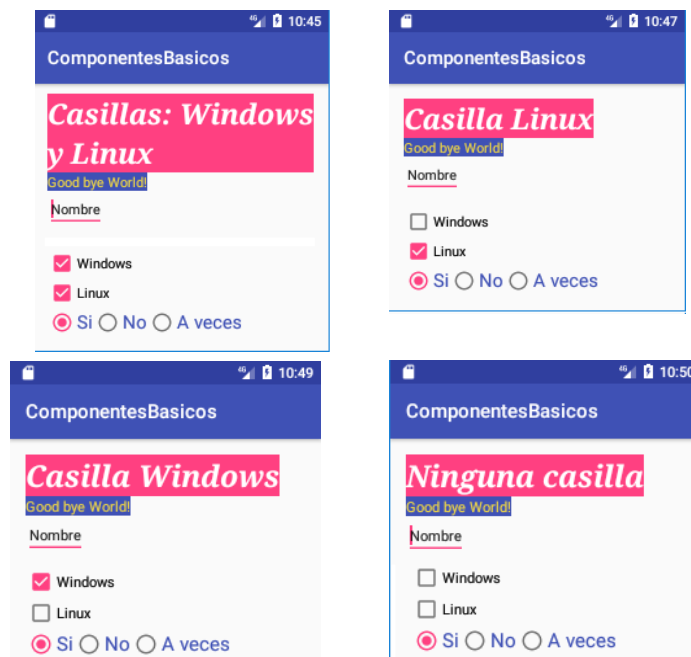
```
<CheckBox
    android:id="@+id/chk_2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Linux" /> (no existe propiedad android:onClick)
```

```
chk_2.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton compoundButton, boolean b) {
        if(b)
            tv_1.setText("Chequeado Linux");
        else
            tv_1.setText("Desmarcado Linux");
    }
});
```

- La variable booleana recibida como parámetro indica el nuevo estado de la casilla de verificación (*true*: seleccionada; *false*: no seleccionada).

- **Propuesta :**

Modificar el código de tal forma que se detecten las diferentes posibilidades en cuanto a la selección de las dos casillas de verificación existentes, como muestran las siguientes capturas:



3.4. RadioButton

- Funciona de forma similar a CheckBox.
- Se puede comprobar el estado de un botón mediante el método **isChecked()**.
- También se puede hacer uso de la propiedad **android:onClick**.
- Ejemplo:

```
<RadioGroup
    android:id="@+id/rg1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <RadioButton
        android:id="@+id/rb1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Si"
        android:checked="true"
        android:onClick="chequearRadioButton"/>
    <RadioButton
        android:id="@+id/rb2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="No"
        android:onClick="chequearRadioButton"/>
</RadioGroup>
```

- Para obtener el RadioButton seleccionado se puede hacer de diferentes formas:

```
public void chequearRadioButton (View v){
    if (rb1.isChecked())
        tv_1.setText("Has seleccionado Sí");
    else
        tv_1.setText("Has seleccionado No");
}
```

O también:

```
private View.OnClickListener mListener = new View.OnClickListener() {
    @Override
    public void onClick(View v){
        if (v.getId() == R.id.rb1)
            tv_1.setText("Has seleccionado Sí");
        else
            tv_1.setText("Has seleccionado No");
    } // end onClick
};
```

- También es posible obtener el identificador del botón de radio seleccionado en un grupo de botones (RadioGroup) mediante el método **getCheckedRadioButtonId()**, aplicado al objeto de tipo RadioGroup:

```
public void chequearRB(View v){
    if(rg_1.getCheckedRadioButtonId()==R.id.rb1)
        tv_1.setText("Seleccionado SI");
    else if (rg_1.getCheckedRadioButtonId()==R.id.rb2)
        tv_1.setText("Seleccionado NO");
    else tv_1.setText("Seleccionado A VECES");
}
```

4. VISIBILIDAD DE DETERMINADOS ELEMENTOS

- Como consecuencia de la gestión de un evento puede ser interesante la ocultación de una o varias vistas del layout, así como la aparición de vistas nuevas.
- En el archivo de layout, a las Views se les puede añadir el atributo **android:visibility**, para hacer que salgan o no en pantalla en el momento de la carga de dicho layout.
- La propiedad **android:visibility** puede tener tres valores:
 - **Visible**. La View se muestra en pantalla en la posición que le corresponde. Es el valor por defecto.
 - **Invisible**. El elemento se sitúa en la posición que le corresponda en la pantalla, no se muestra pero guarda su sitio. Es decir, si tenemos en un LinearLayout vertical tres botones y al del medio se le pone el atributo como “invisible”, se verá un espacio en blanco entre el primer y el tercer botón.
 - **Gone**. Es parecido al anterior pero, en este caso, el elemento no reserva su espacio. En el ejemplo de los tres botones, si se pusiera el atributo del segundo botón como “gone”, el efecto sería que el primer y tercer botón aparecerían uno seguido del otro, sin guardar espacio para el segundo
- La visibilidad se puede modificar posteriormente, por código, mediante el método **setVisibility(int)**. El valor del parámetro de tipo entero se suele indicar con una de las tres constantes: **View.VISIBLE**, **View.INVISIBLE** y **View.GONE**.
- Elemento **<ScrollView>**.
 - Al modificar la visibilidad de algunos elementos puede surgir un problema, y es que, al mostrar los elementos que antes estaban ocultos, no quepan en la pantalla y se “pierda” alguno de ellos.
 - Para solucionarlo, podemos usar un nuevo elemento en el Layout. Se trata del elemento **<ScrollView>**, que es una vista que puede ser más grande que la propia pantalla, y que se puede deslizar para ver su contenido.
 - El elemento **<ScrollView>** sólo puede tener un elemento hijo, pero este puede ser todo lo complejo que se necesite.