

# Módulo 8: Interfaces de Usuario

José Francisco Chicano García

Departamento de Lenguajes y  
Ciencias de la Computación  
Universidad de Málaga



Samsung  
**TECH INSTITUTE**

# Datos de Contacto



3.2.47, E.T.S.I. Informática



chicano@lcc.uma.es



@francischicano

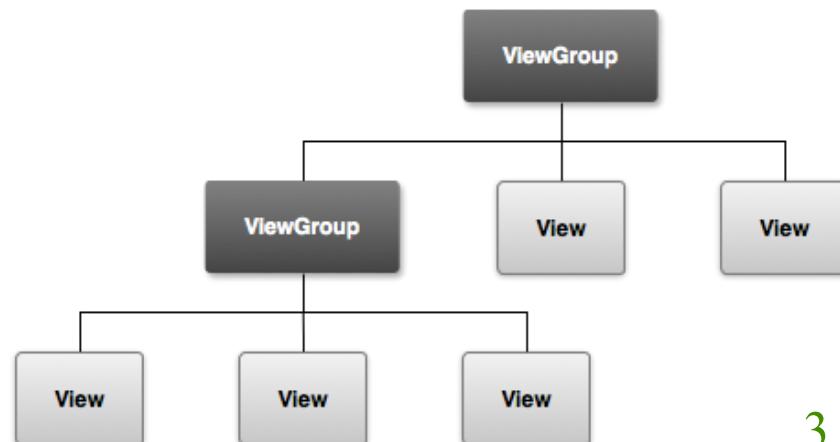


www.franciscochicano.es

# Introducción a la GUI

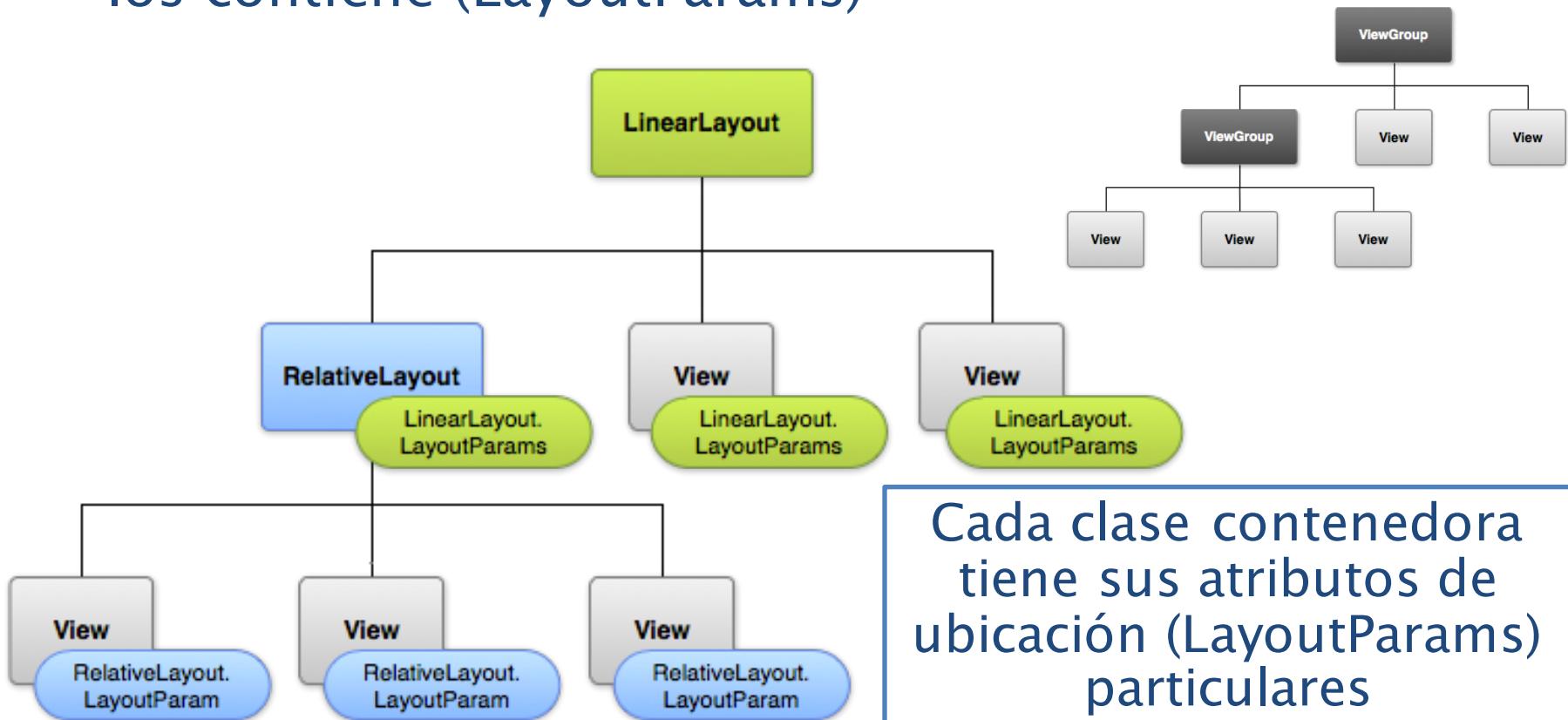
- Podemos clasificar todos los elementos visuales de la GUI de Android en dos categorías:
  - Contenedores: heredan de **ViewGroup**
  - Componentes: heredan de **View**
- Desde el punto de vista de la OO los contenedores (**ViewGroup**) heredan de los componentes (**View**)
- Los contenedores pueden “contener” en su interior otros componentes (incluyendo contenedores)

Esta organización permite estructurar la GUI y reutilizar componentes



# Introducción a la GUI

- Los componentes tienen asociados atributos que indican cómo se van a ubicar en el contenedor que los contiene (LayoutParams)



# Introducción a la GUI

- Hay esencialmente tres formas de crear las vistas:
  - De forma programada: se crean objetos de las clases que representan los objetos gráficos, se añaden a los componentes y se establecen todas las propiedades relacionadas con la ubicación y aspecto (no lo haremos en este curso)
  - Editando directamente un fichero XML que se guarda en res/layout. En tiempo de ejecución, la API de Android toma el XML, lo procesa y construye los objetos gráficos que en él se describen (ya se ha visto en módulos anteriores).
  - Usando el editor de vistas que trae Android Studio. Este editor modifica el fichero XML (nos centraremos en esta alternativa).

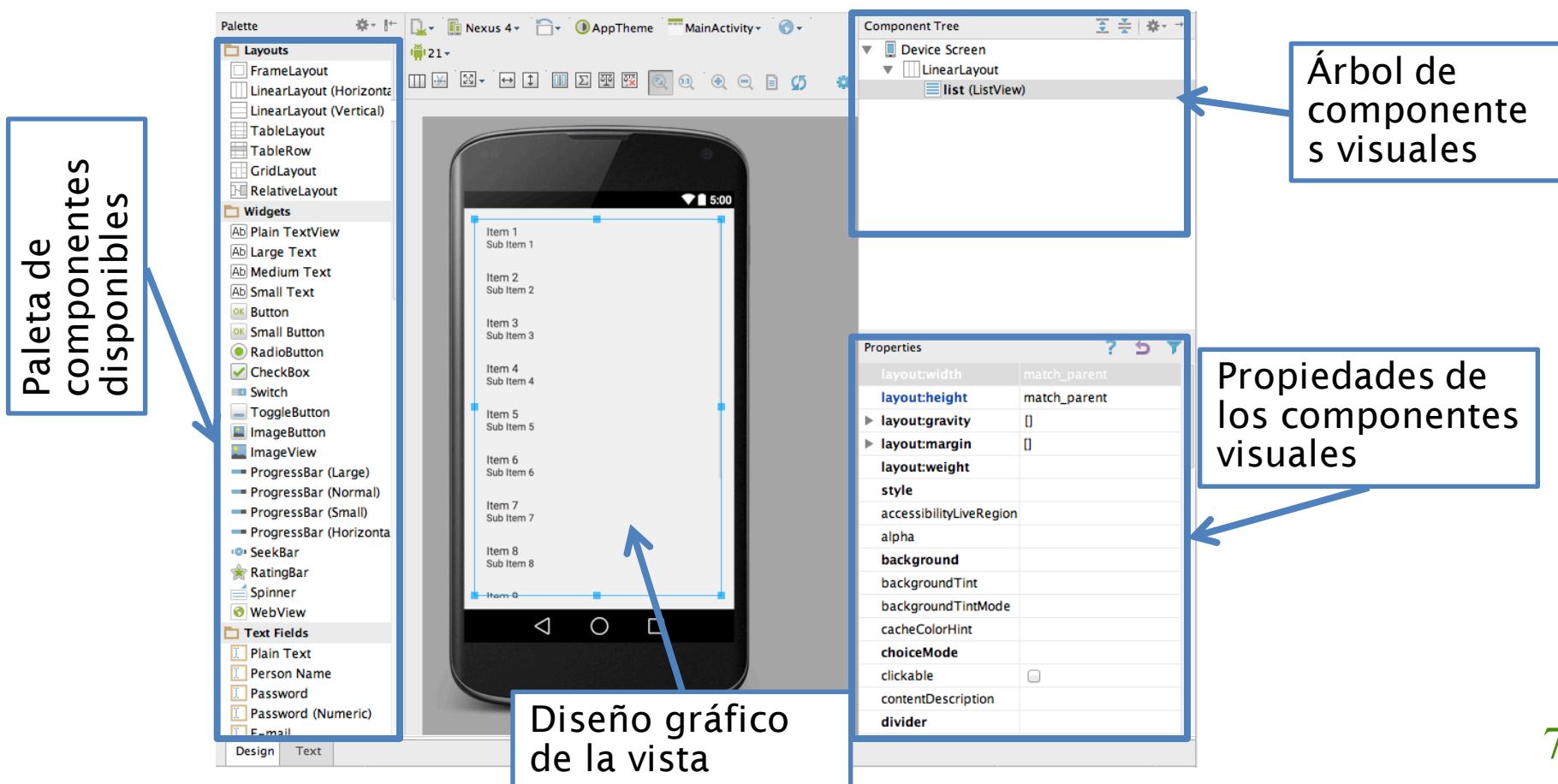
# Introducción a la GUI

- Ejemplo de fichero XML

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:orientation="vertical"
4     android:layout_width="fill_parent"
5     android:layout_height="fill_parent"
6     >
7 <TextView
8     android:layout_height="wrap_content"
9     android:text="@string/hello"
10    android:layout_gravity="center_horizontal"
11    android:layout_width="wrap_content"/>
12 <EditText
13     android:layout_width="fill_parent"
14     android:layout_height="wrap_content"
15     android:id="@+id/texto"/>
16 <Button
17     android:layout_width="wrap_content"
18     android:layout_height="wrap_content"
19     android:id="@+id/boton"
20     android:layout_gravity="center_horizontal"
21     android:text="@string/boton"/>
22 </LinearLayout>
23
```

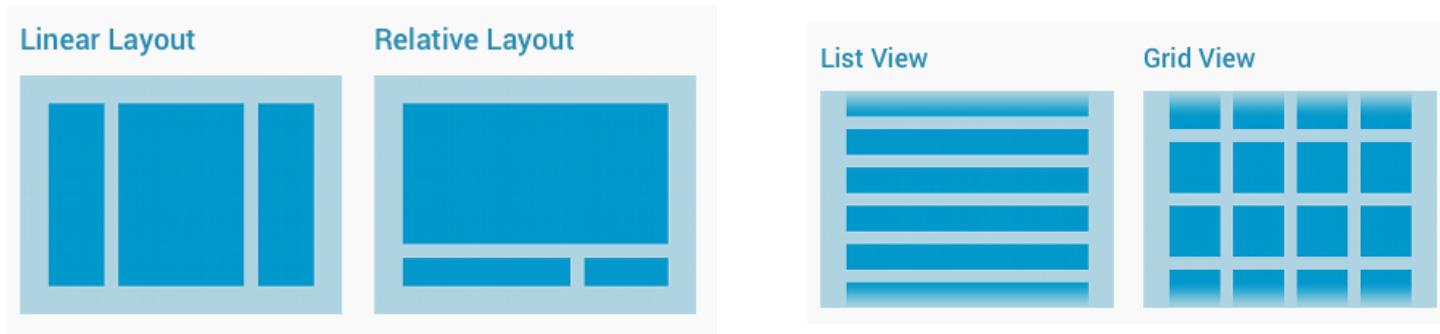
# Diseño de GUI con Android Studio

- Editor de Android Studio. Permite arrastrar los componentes y asignarles los parámetros de ubicación fácilmente



# Diseño de GUI con Android Studio

- A la hora de diseñar una GUI, escogeremos en primer lugar el contenedor raíz más apropiado para nuestra aplicación en función de cómo queramos colocar los componentes visuales

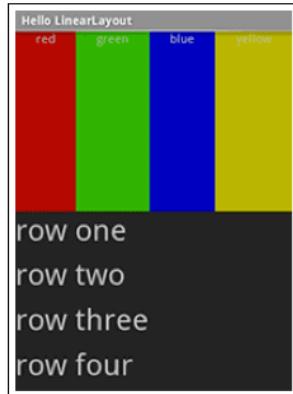


- **RelativeLayout** es muy flexible y posiblemente el mejor para la mayoría de los casos

# Diseño de GUI con Android Studio

- Algunos ejemplos de contenedores con componentes visuales ya ubicados

Linear Layout



Relative Layout

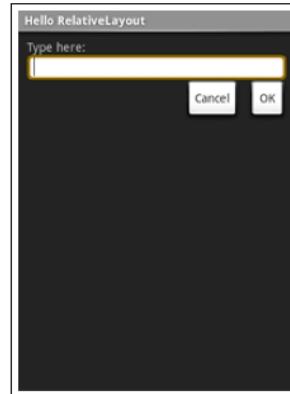
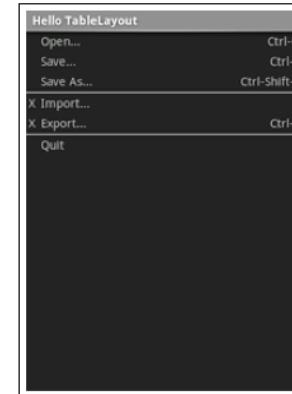
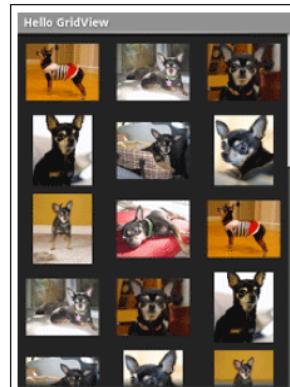


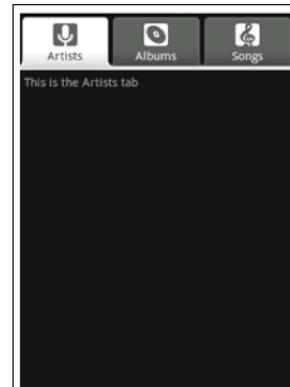
Table Layout



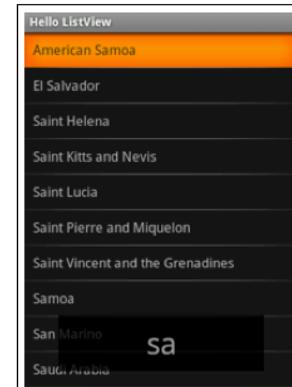
Grid View



Tab Layout

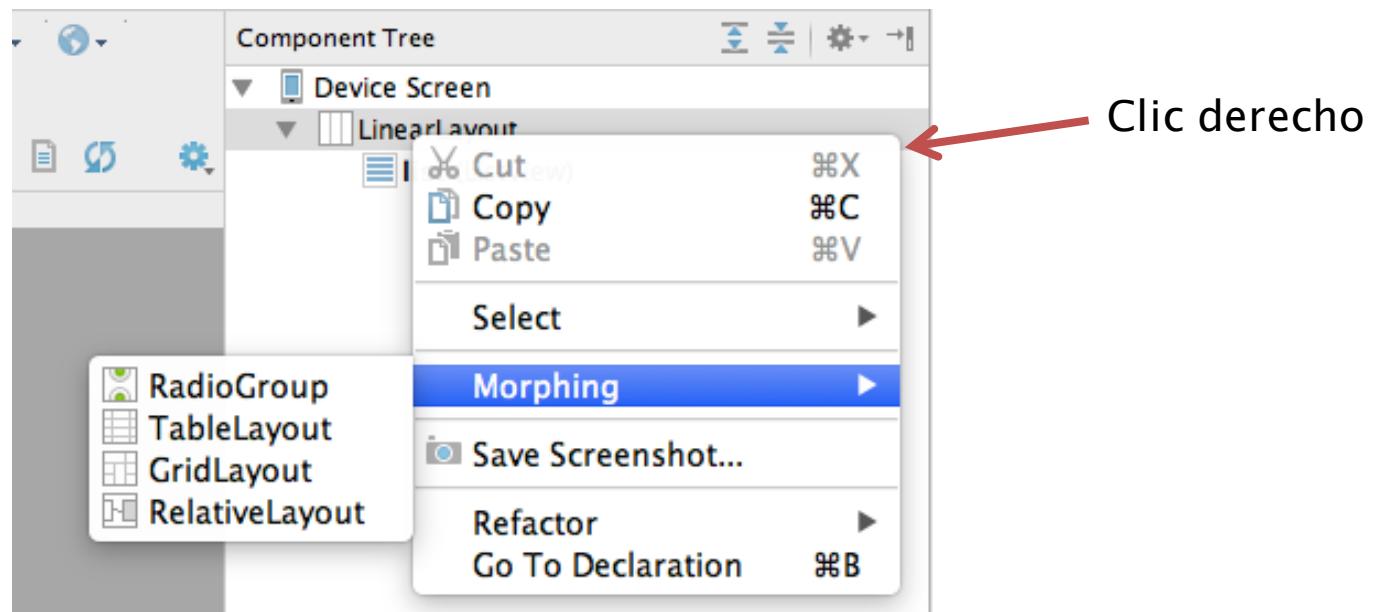


List View



# Diseño de GUI con Android Studio

- Si el contenedor que pretendemos utilizar no es el que hay en nuestro diseño actual lo podemos cambiar haciendo clic con el botón derecho del ratón en el contenedor dentro el árbol de componentes y escogiendo la opción “Morphing”
- Si esta opción no aparece, debemos editar el fichero XML o crear un fichero de layout nuevo



# Diseño de GUI con Android Studio

- **Ejercicio 8.1**

- Crea un proyecto Android usando el asistente y dejando todos los valores por defecto
- Aparecerá una vista inicial con un TextView contenido en un RelativeLayout
- Cambia el contenedor a LinearLayout (horizontal)
- Añade dos botones: uno a la derecha y otro a la izquierda del texto



# Diseño de GUI con Android Studio

- **Ejercicio 8.2**

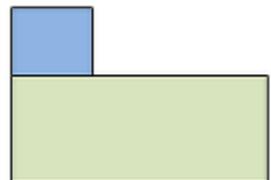
- Vamos a editar algunas de las propiedades de ubicación relacionadas con el botón de más a la izquierda
- Establece la propiedad de gravedad (gravity) al valor “bottom” ¿Qué pasa?
- Asigna al margen de la derecha (margin right) un valor 30dp ¿Qué ha ocurrido ahora?
- Asigna al margen de la izquierda (margin left) un valor de 30 dp ¿Qué ha cambiado?
- Establece la gravedad del botón de la derecha al valor “center\_vertical” ¿Qué ocurre con ese botón?
- Asigna al TextView un margen superior (margin top) de 30 dp ¿Hacia dónde se desplaza el texto?

# Diseño de GUI con Android Studio

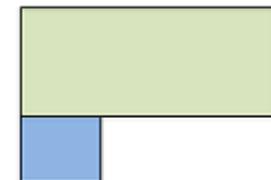
- En los ejercicios anteriores has practicado un poco con las propiedades de ubicación (layout params) del contenedor LinearLayout
- El contenedor RelativeLayout es más flexible, ya que permite alinear cada componente con los bordes del contenedor o los bordes de otros componentes
- Para ello dispone de muchas más propiedades de ubicación que LinearLayout
- Veamos las más importantes y su significado

# Diseño de GUI con Android Studio

- Propiedades de alineación entre componentes visuales



android:layout\_above="base"



android:layout\_below="base"



android:layout\_alignTop="base"



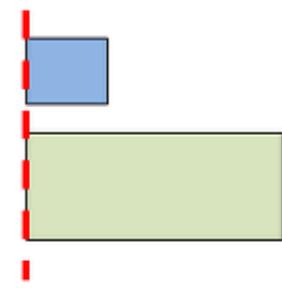
android:layout\_toLeftOf="base"



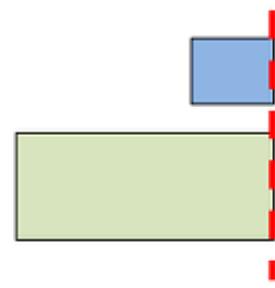
android:layout\_toRightOf="base"



android:layout\_alignBottom="base"



android:layout\_alignLeft="base"



android:layout\_alignRight="base"

# Diseño de GUI con Android Studio

- Propiedades de alineación con el contenedor



android:layout\_alignParentLeft



android:layout\_alignParentTop



android:layout\_alignParentRight



android:layout\_alignParentBottom



android:layout\_centerHorizontal



android:layout\_centerVertical

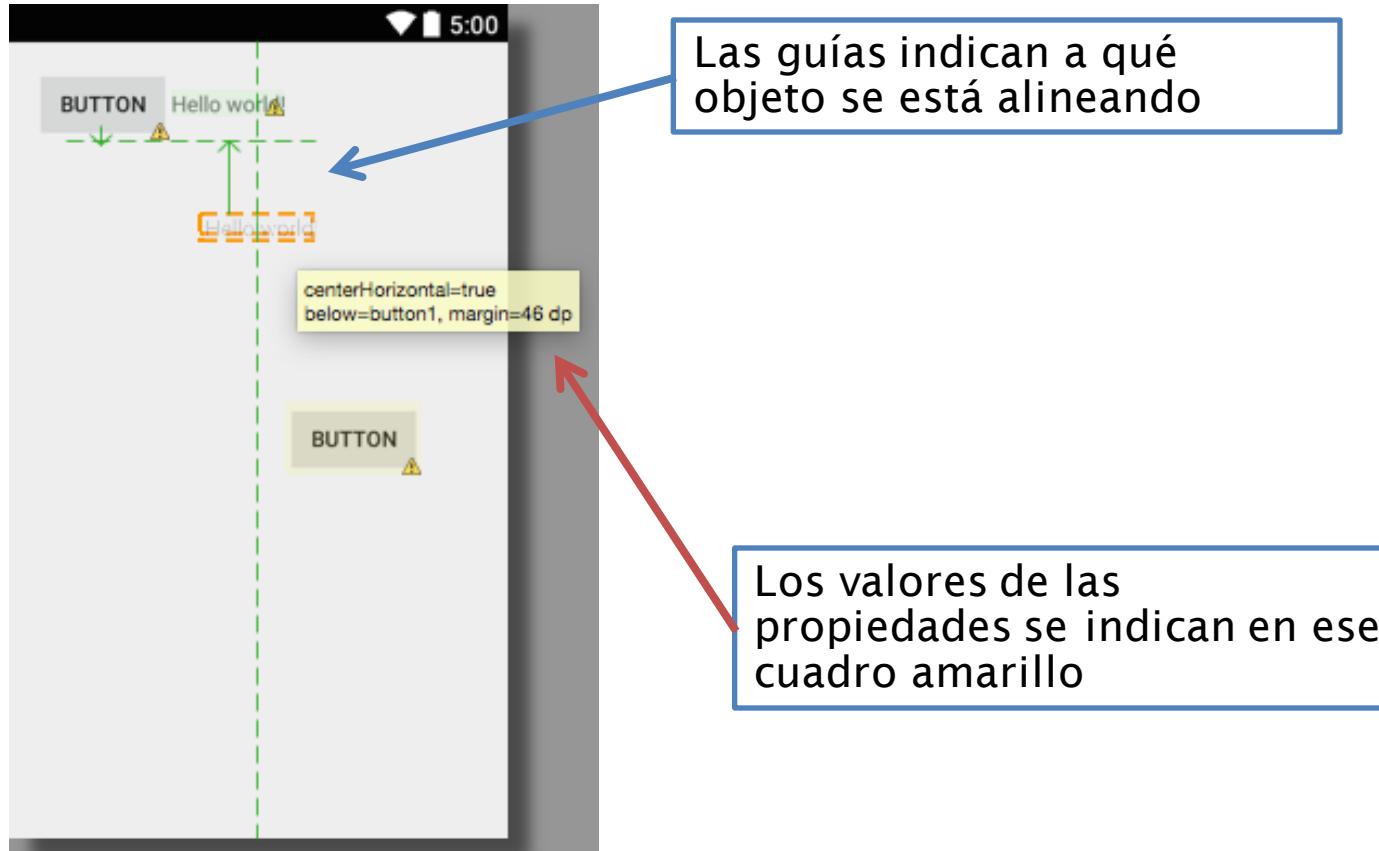


android:layout\_centerInParent

Estas propiedades se pueden combinar con las que vimos anteriormente y además se pueden usar márgenes para separar componentes

# Diseño de GUI con Android Studio

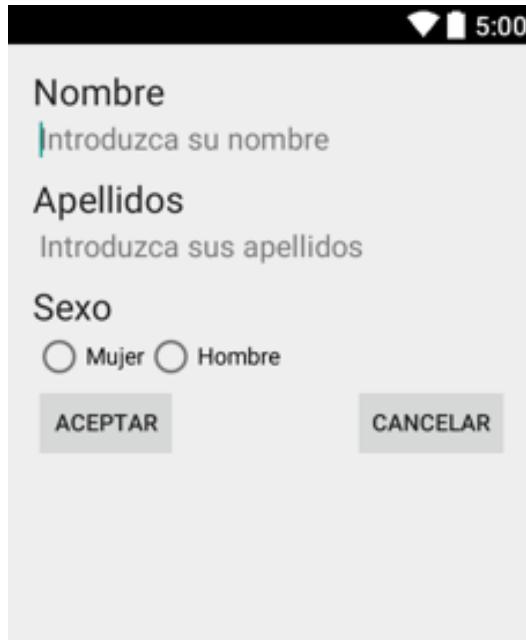
- Estas propiedades pueden establecerse en el panel de propiedades pero también pueden modificarse arrastrando los componentes



# Diseño de GUI con Android Studio

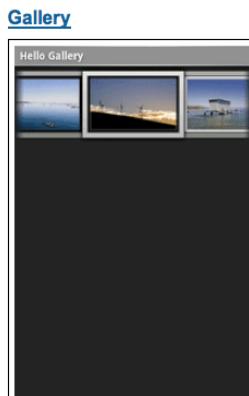
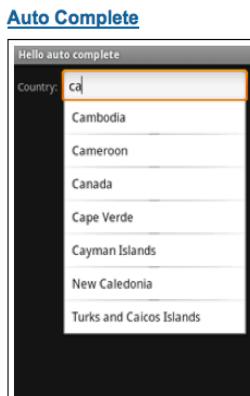
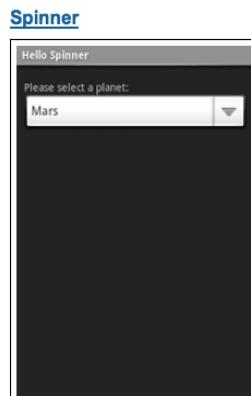
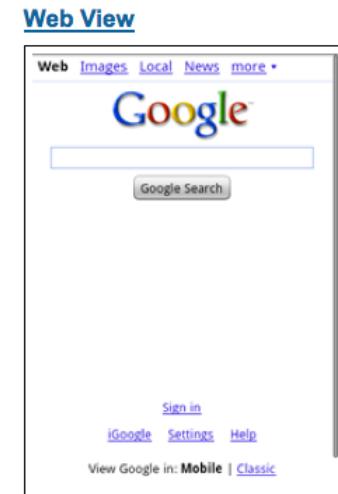
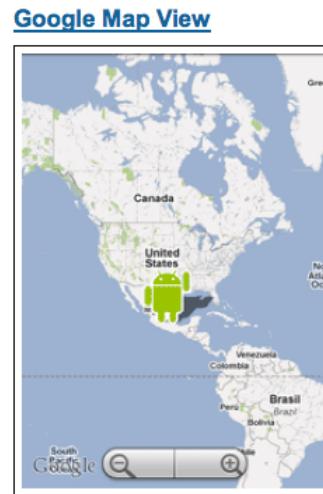
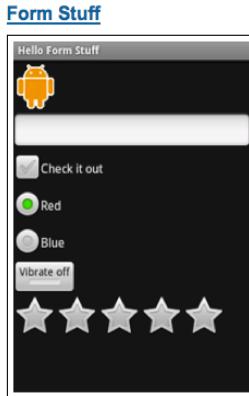
- **Ejercicio 8.3**

- Crea un proyecto Android usando el asistente y dejando todos los valores por defecto
- Aparecerá una vista inicial con un TextView contenido en un RelativeLayout
- Elimina el TextView para dejar el diseño en blanco
- Añade y alinea los componentes visuales para que queden como en la figura



# Diseño de GUI con Android Studio

- Despues de elegir el contenedor raíz más apropiado debemos incluir los componentes visuales en el contenedor
- A continuación se muestran algunos de estos componentes



Estos componentes se pueden encontrar en la paleta de componentes del editor de GUIs

# Diseño de GUI con Android Studio

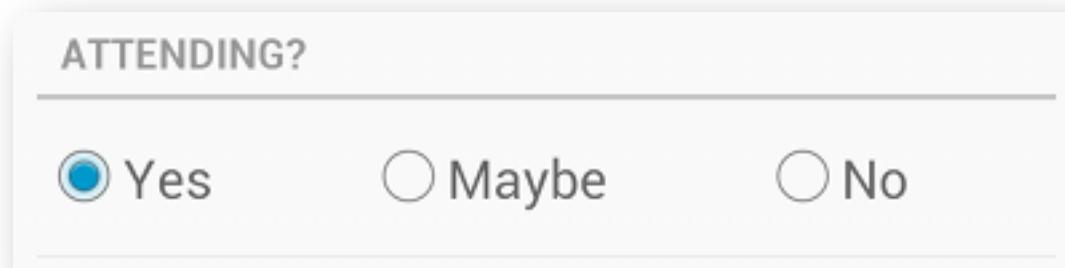
- Veamos algunos de los componentes más importantes

- TextView
- EditText
- Button
- RadioButton
- CheckBox
- Toggle buttons y Switches
- ProgressBar
- SeekBar
- GridView
- ListView
- Spinner

Ya vistos en el módulo 7

# Diseño de GUI con Android Studio

- **RadioButton**
- Son botones que tienen dos estados pulsados o no
- Suelen representar opciones excluyentes
- Cuando el usuario pulsa uno de ellos se selecciona
- El botón que estaba seleccionado hasta ese momento se deselecciona



# Diseño de GUI con Android Studio

- **RadioButton**
- Para que solo pueda existir un botón seleccionado entre un grupo de botones se deben incluir en un contenedor de clase **RadioGroup** (que es una subclase de **LinearLayout**)
- Para comprobar si un botón concreto está pulsado podemos usar el método **isChecked()** que devuelve **true** si el botón está pulsado y **false** en caso contrario
- Si queremos recibir notificaciones cuando se produce un cambio de estado podemos añadir un listener de clase **OnCheckedChangeListener**

## Public Methods

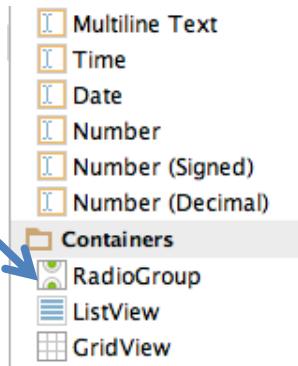
abstract void	<b>onCheckedChanged(CompoundButton buttonView, boolean isChecked)</b>
Called when the checked state of a compound button has changed.	

# Diseño de GUI con Android Studio

- **RadioButton**
- ¿Cómo añadir botones de radio a nuestra vista?

RadioGroup

Así podemos estar informados de los cambios de estado

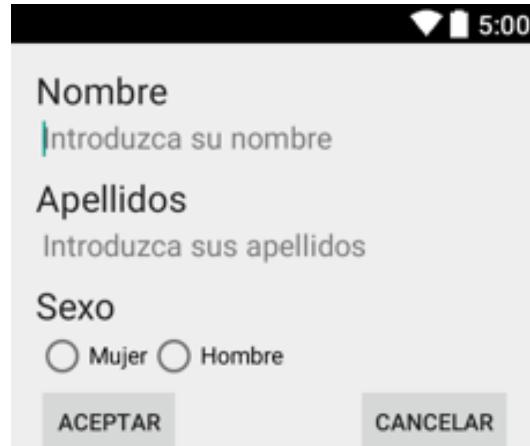


```
radio0.setOnCheckedChangeListener(new OnCheckedChangeListener() {  
    @Override  
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {  
        Log.v(TAG, "Cambió el estado a "+(isChecked?"":"no ")+"pulsado");  
    }  
});
```

# Diseño de GUI con Android Studio

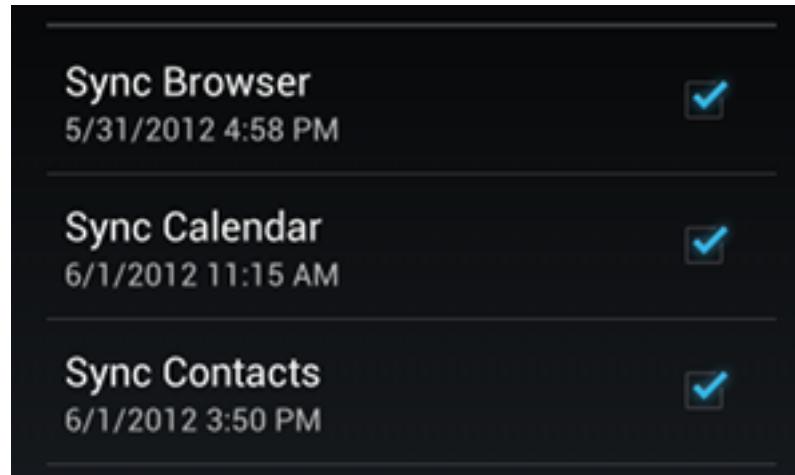
- **Ejercicio 8.4**

- Modifica el ejercicio 8.3 para que los botones de radio relacionados con el sexo estén agrupados (sean excluyentes).
- Al arrancar la aplicación el botón “Mujer” debe estar pulsado y el botón “Hombre” no pulsado.
- Haz que cada vez que el usuario cambie el estado de cualquiera de los botones relacionados con el sexo aparezca un “Toast” indicando el nuevo valor seleccionado.
- Cuando el usuario pulsa “Aceptar” la aplicación debe mostrar otra actividad en la que se presentan en una etiqueta los valores de cada campo (nombre, apellidos y sexo).
- Si el usuario pulsa “Cancelar” la aplicación termina.



# Diseño de GUI con Android Studio

- **CheckBox**
- Son botones que tienen dos estados pulsados o no
- Suelen representar opciones no excluyentes
- Cuando el usuario pulsa uno de ellos se selecciona
- A diferencia de los **RadioButton** no se agrupan y puede haber varios Checkboxes pulsados simultáneamente



# Diseño de GUI con Android Studio

- **CheckBox**
- Para comprobar si un botón concreto está pulsado podemos usar el método `isChecked()` que devuelve `true` si el botón está pulsado y `false` en caso contrario
- Si queremos recibir notificaciones cuando se produce un cambio de estado podemos añadir un listener de clase `OnCheckedChangeListener`

## Public Methods

abstract void

`onCheckedChanged(CompoundButton buttonView, boolean isChecked)`

Called when the checked state of a compound button has changed.

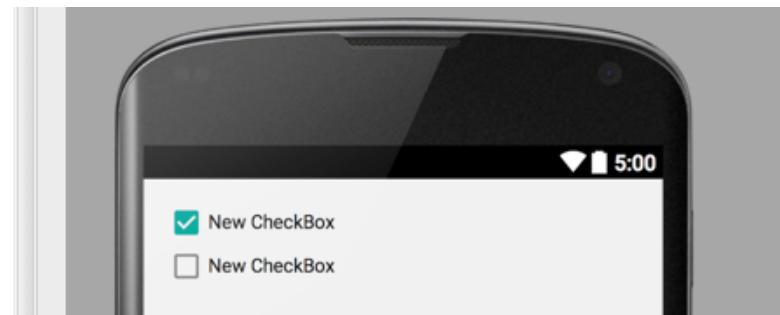
# Diseño de GUI con Android Studio

- CheckBox
- ¿Cómo añadir checkboxes a nuestra vista?

Checkbox

Así podemos estar informados de los cambios de estado

- Ab Large Text
- Ab Medium Text
- Ab Small Text
- OK Button
- OK Small Button
- RadioButton
- CheckBox
- Switch
- ToggleButton



```
checkbox1.setOnCheckedChangeListener(new OnCheckedChangeListener() {  
    @Override  
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {  
        Log.v(TAG, "Cambió el estado a "+(isChecked?"":"no ")+"pulsado");  
    }  
});
```

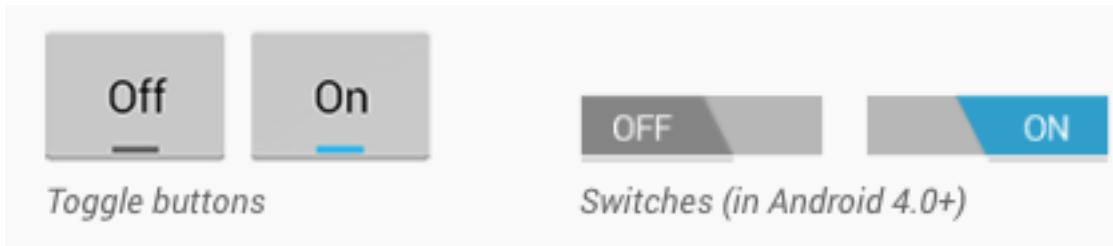
# Diseño de GUI con Android Studio

- **Ejercicio 8.5**

- Modifica el ejercicio 8.4 para añadir un Checkbox que registre si el usuario es fumador o no.
- Al arrancar la aplicación el Checkbox debe estar vacío.
- Haz que cada vez que el usuario cambie el estado del checkbox aparezca un “Toast” indicando si está pulsado o no.
- Cuando el usuario pulsa “Aceptar” la aplicación debe mostrar otra actividad en la que se presentan en una etiqueta los valores de cada campo (nombre, apellidos, sexo y fumador o no).
- Si el usuario pulsa “Cancelar” la aplicación termina.

# Diseño de GUI con Android Studio

- **Toggle buttons y switches**
- Al igual que los checkboxes tienen dos estados
- La diferencia entre un toggle button, un switch y un checkbox es simplemente gráfica



- Para comprobar si el botón está activo podemos usar el método `isChecked()` que devuelve `true` si el botón está pulsado y `false` en caso contrario
- Si queremos recibir notificaciones cuando se produce un cambio de estado podemos añadir un listener de clase `OnCheckedChangeListener`

# Diseño de GUI con Android Studio

- **Toggle buttons y switches**
- ¿Cómo añadir estos componentes a nuestra vista?

The screenshot shows the Android Studio interface with the 'Widgets' palette open. The palette lists various UI components: Plain TextView, Large Text, Medium Text, Small Text, Button, Small Button, RadioButton, CheckBox, Switch, ToggleButton, and ImageButton. Two arrows point from the text 'Switch' and 'Toggle button' to their respective entries in the palette. A callout box with the text 'Así podemos estar informados de los cambios de estado' points to the code snippet below.

Switch

Toggle button

Así podemos estar informados de los cambios de estado

```
toggleButton.setOnCheckedChangeListener(new OnCheckedChangeListener() {  
    @Override  
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {  
        Log.v(TAG, "Cambió el estado a "+(isChecked?"" :"no "+) +"pulsado");  
    }  
});
```

# Diseño de GUI con Android Studio

- **Ejercicio 8.6**
  - Sustituye el checkbox del ejercicio 8.5 por un switch.

# Diseño de GUI con Android Studio

- **ProgressBar**
- Es un componente versátil que nos permite indicar el progreso de una operación “lenta” o mostrar una animación mientras se espera que ocurra un evento.
- Cuando estamos esperando un evento la barra de progreso tiene un valor indeterminado y se muestra así:



- Cuando tiene un valor determinado la barra muestra el siguiente aspecto:



# Diseño de GUI con Android Studio

- **ProgressBar**
- El tipo de barra de progreso (determinado o indeterminado) puede consultarse con el método:
  - `boolean isIndeterminate()`
- y puede modificarse con
  - `void setIndeterminate(boolean indeterminate)`
- En el caso de que la barra de progreso sea determinada se puede establecer el valor de progreso con:
  - `void setProgress(int progress)`
- el valor puede consultarse con:
  - `int getProgress()`
- Por defecto el progreso es un número entre 0 y 100, pero el valor máximo puede consultarse y modificarse con `get/setMax()`

# Diseño de GUI con Android Studio

- **ProgressBar**
- Cuando queramos hacer desaparecer la barra de progreso podemos poner su visibilidad a GONE:

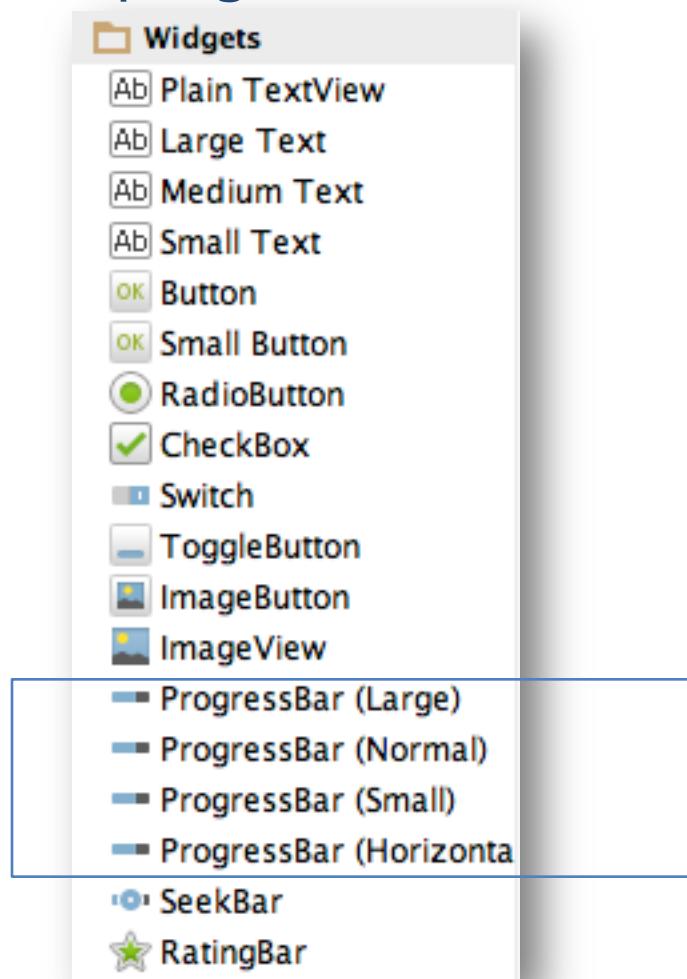
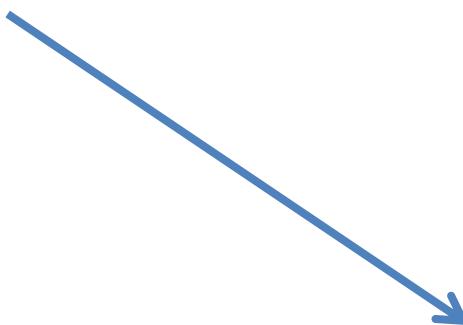
```
progreso.setVisibility(View.GONE);
```

- Pero debemos tener cuidado si la barra de progreso tiene relación con otros componentes (en un RelativeLayout, por ejemplo)
- Además de setProgress, se puede aumentar el valor de progreso con el método **incrementProgressBy()**

# Diseño de GUI con Android Studio

- **ProgressBar**
- ¿Cómo añadir barras de progreso a nuestra vista?

ProgressBar



# Diseño de GUI con Android Studio

- **Ejercicio 8.7**

- Desarrolla una aplicación Android que contenga una barra de progreso de estilo indeterminado, una barra de estilo determinado y un botón.
- Cada vez que el usuario pulse el botón la barra de estilo determinado debe aumentar su progreso en 20 unidades

# Diseño de GUI con Android Studio

- **SeekBar**
- La “barra de búsqueda” permite al usuario introducir un valor entero dentro de un rango de posibles valores.
- Hereda de ProgressBar y por tanto, los métodos para consultar y establecer el valor de progreso son los mismos: **get/setProgress(int progreso)**
- Podemos ser notificados cuando se produce un cambio en el valor de progreso.
- Para ello se utiliza un objeto de clase **OnSeekBarChangeListener**:

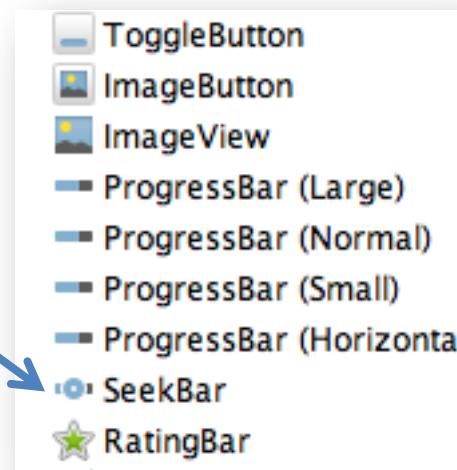
Public Methods	
abstract void	<b>onProgressChanged(SeekBar seekBar, int progress, boolean fromUser)</b> Notification that the progress level has changed.
abstract void	<b>onStartTrackingTouch(SeekBar seekBar)</b> Notification that the user has started a touch gesture.
abstract void	<b>onStopTrackingTouch(SeekBar seekBar)</b> Notification that the user has finished a touch gesture.

# Diseño de GUI con Android Studio

- **SeekBar**
- ¿Cómo añadir barras de búsqueda a nuestra vista?

Así podemos estar informados de los cambios de valor

SeekBar



```
seekbar.setOnSeekBarChangeListener(new OnSeekBarChangeListener() {  
    @Override  
    public void onProgressChanged(SeekBar seekBar, int progress,  
        boolean fromUser) {  
        etiqueta.setText(String.valueOf(progress));  
    }  
}
```

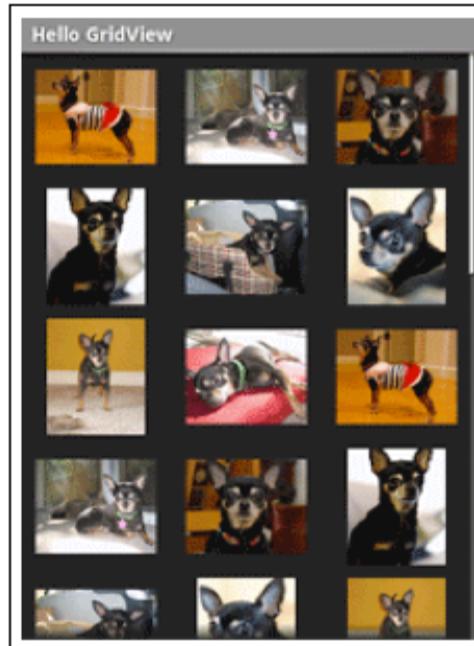
# Diseño de GUI con Android Studio

- **Ejercicio 8.8**

- Desarrolla una aplicación Android que contenga una barra de búsqueda y una etiqueta.
- Cada vez que el usuario modifique el valor de la barra, la etiqueta deberá mostrar dicho valor.

# Diseño de GUI con Android Studio

- **GridView**
- Es un contenedor que permite mostrar información (fotos, texto, botones, etc.) organizada en una rejilla.



- Posee propiedades para controlar el número de columnas o su tamaño, la separación horizontal y vertical, etc.

# Diseño de GUI con Android Studio

- **GridView**
- Es una subclase de **AdapterView** (como ListView y Spinner, que veremos más adelante).
- Todos los **AdapterView** funcionan tomando la información a mostrar de un “adaptador”.
- Un “adaptador” es un objeto que proporciona la información a mostrar y la forma en que se debe mostrar.
- Android proporciona algunos adaptadores básicos para mostrar texto pero si queremos más flexibilidad (por ejemplo mostrar imágenes) debemos implementar nuestro propio adaptador.

# Diseño de GUI con Android Studio

- **GridView**
- Una vez que hemos implementado (o reutilizado) un adaptador para nuestro **GridView**, informamos al objeto GridView de cuál es su adaptador mediante el método **setAdapter**

```
GridView gridview = (GridView) findViewById(R.id.gridview);
gridview.setAdapter(new ImageAdapter(this));
```

- Cuando hacemos esto, el GridView consultará el adaptador para averiguar qué información debe mostrar.
- Un adaptador muy útil que Android proporciona es **ArrayAdapter<T>** que representa una lista (o array) de objetos de clase T arbitraria.
- A este adaptador se pueden añadir objetos con **add** y eliminar con **remove**.

# Diseño de GUI con Android Studio

- **GridView**
- ArrayAdapter posee 6 constructores, cuyos parámetros se pueden explicar teniendo en cuenta dos aspectos:
  - Los objetos que van a formar parte inicialmente del adaptador

Public Constructors	
	ArrayAdapter(Context context, int resource) Constructor
	ArrayAdapter(Context context, int resource, int textViewResourceId) Constructor
	ArrayAdapter(Context context, int resource, T[] objects) Constructor
	ArrayAdapter(Context context, int resource, int textViewResourceId, T[] objects) Constructor
	ArrayAdapter(Context context, int resource, List<T> objects) Constructor
	ArrayAdapter(Context context, int resource, int textViewResourceId, List<T> objects) Constructor

Diagrama que agrupa los constructores de ArrayAdapter en tres categorías basadas en los datos que se pasan:

- Los tres primeros constructores (que no pasan una lista) están agrupados por un paréntesis abierto y cerrado y la etiqueta "No se dan los datos".
- Los tres últimos constructores (que pasan una lista) están agrupados por un paréntesis abierto y cerrado y la etiqueta "Se dan en una lista".
- Los tres primeros constructores y sus respectivas categorías están agrupados por un paréntesis abierto y cerrado y la etiqueta "Se dan en un array".

# Diseño de GUI con Android Studio

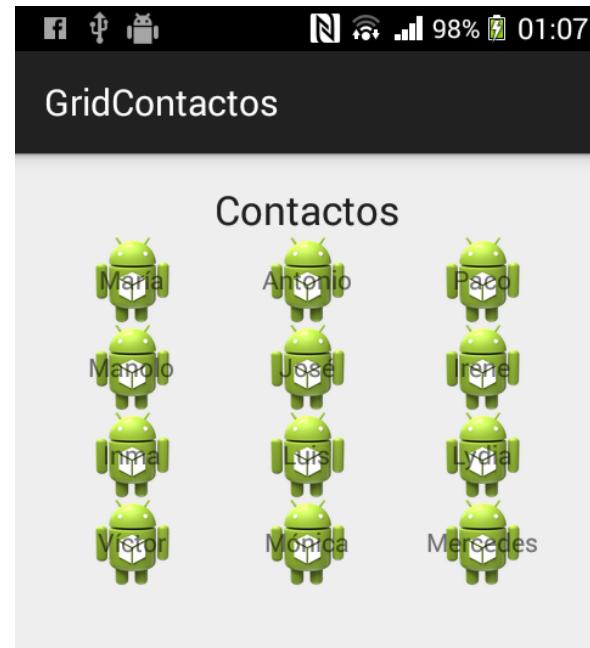
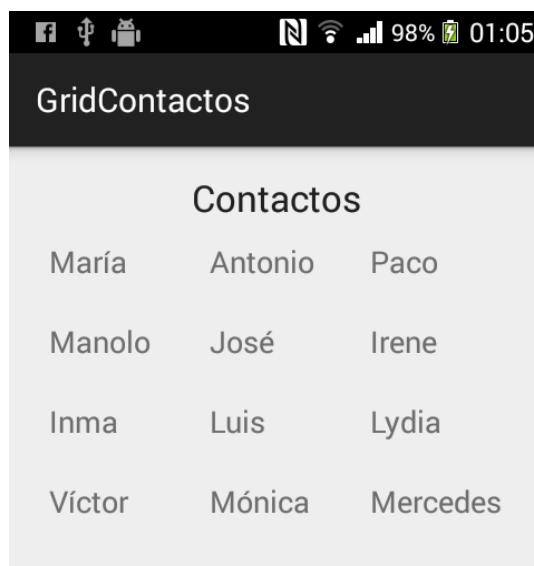
- **GridView**
- ArrayAdapter posee 6 constructores, cuyos parámetros se pueden explicar teniendo en cuenta dos aspectos:
  - El aspecto que van a tener los datos al ser mostrados: ArrayAdapter SIEMPRE muestra los datos como texto y para obtener este texto invoca el método `toString()` de los objetos.
    - Cuando solo hay un argumento entero, este es el identificador de un recurso de tipo Layout que debe contener exactamente un TextView como raíz
    - Cuando hay dos argumentos enteros, el primero es el identificador del recurso de tipo Layout que se usará para mostrar los datos y el segundo debe ser un identificador de un TextView que está contenido en dicho layout y que será donde se muestre el texto.

Public Constructors	
	<code>ArrayAdapter(Context context, int resource)</code> Constructor
	<code>ArrayAdapter(Context context, int resource, int textViewResourceId)</code> Constructor

# Diseño de GUI con Android Studio

- **Ejercicio 8.9**

- Desarrolla una aplicación Android que muestre en un GridView los siguientes nombre (supuestos contactos): María, Antonio, Paco, Manolo, José, Irene, Inma, Luis, Lydia, Víctor, Mónica, Mercedes.
- A continuación modifica la aplicación para que el nombre aparezca delante de un icono de un androide.



# Diseño de GUI con Android Studio

- **GridView**
- Si lo que queremos es mostrar información que no es textual (imágenes, por ejemplo) tenemos que crear nuestro propio adaptador.
- Todo adaptador tiene un método **getView** que devuelve un componente visual que se usará para representar el objeto de una determinada posición.
- Si queremos devolver un componente visual arbitrario en cada posición debemos sobreescribir este método.

# Diseño de GUI con Android Studio

- **GridView**
- El siguiente código se corresponde con un adaptador basado en un array de objetos “Drawable” (pintables) que devuelve imágenes.

```
public class ImageAdapter extends ArrayAdapter<Drawable> {  
    private Context contexto;  
    public ImageAdapter(Context ctx) {  
        super(ctx, 0);  
        contexto=ctx;  
    }  
  
    @Override  
    public View getView(int position, View convertView, ViewGroup parent) {  
        ImageView imagen = new ImageView(contexto);  
        Drawable drawable = getItem(position);  
        imagen.setImageDrawable(drawable);  
        return imagen;  
    }  
}
```

# Diseño de GUI con Android Studio

- **Ejercicio 8.10**
- Con ayuda del código del adaptador anterior, vamos a crear un pequeño álbum de fotos con un GridView.
- La aplicación debe contener un GridView y se debe usar el adaptador cuyo código se ha mostrado en la transparencia anterior.
- Las imágenes se tomarán del campus virtual y se añadirán a la carpeta de recursos “drawable”.
- Para recuperar dichos recursos dentro de la aplicación como objetos de clase Drawable podemos escribir lo que sigue (solo es un ejemplo para un caso particular):

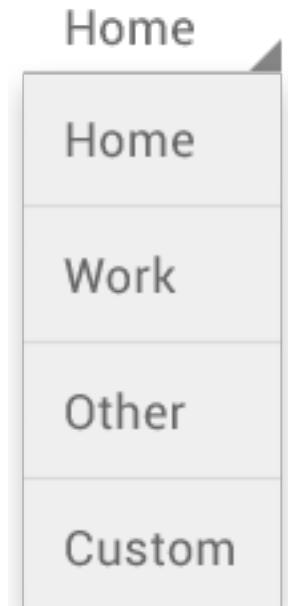
```
Drawable drawable = getDrawable(R.drawable.ic_launcher);
```

Identificador del recurso drawable



# Diseño de GUI con Android Studio

- **Spinner**
- Es un contenedor similar a los denominados “combo box”, que permiten seleccionar una opción de entre un conjunto de opciones.
- Se basan también en un adaptador.
- El adaptador contiene las distintas opciones.
- Este adaptador utiliza dos layouts (vistas):
  - Una para el objeto seleccionado
  - Otra para mostrar las opciones
- Para obtener la segunda, un Spinner utiliza
  - `getDropDownView()`
- En un **ArrayAdapter**, puede usarse `setDropDownViewResource(int)` para establecer esta vista.



# Diseño de GUI con Android Studio

- **Spinner**
- ¿Cómo añadir un Spinner a nuestra vista?

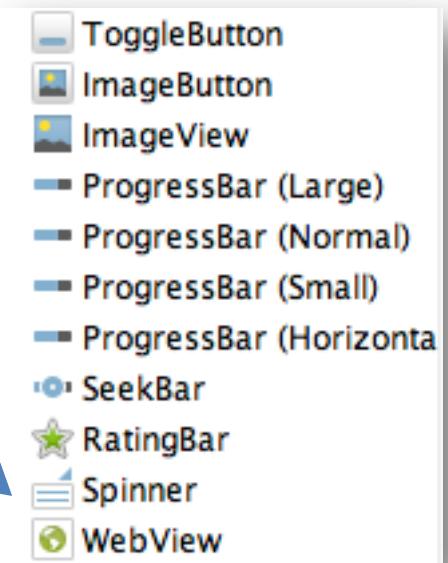
Spinner

Este es un ArrayAdapter  
preparado para un Spinner

Estos layouts los define  
Android y están pensados  
para los Spinner

```
ArrayAdapter<String> adaptador = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item);  
adaptador.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
```

Así se establece el layout para los elelentos en la lista de opciones

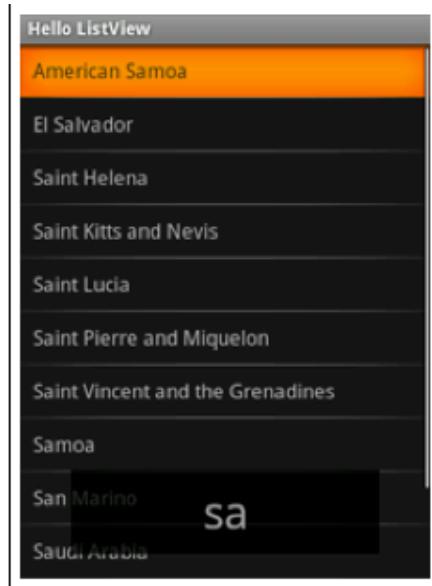


# Diseño de GUI con Android Studio

- **Ejercicio 8.11**
- Modifique la aplicación 8.6 para añadir un campo que pregunte el estado civil de la persona: solter@, casad@, divorciad@, viud@, separad@.

# Diseño de GUI con Android Studio

- **ListView**
- Es un contenedor que permite mostrar información (fotos, texto, botones, etc.) organizada en una lista.



- Se basa en un adaptador del mismo tipo que para **GridView**

# Diseño de GUI con Android Studio

- **ListView**
- Si queremos que en una fila del ListView (o en una celda del GridView) aparezca una layout relativamente complejo es mejor diseñar dicho layout usando el diseñador de Android Studio y luego “inflarlo” cuando se ejecute el programa.
- Para inflar el layout previamente diseñado con Android Studio tenemos que hacer los siguiente en la aplicación (es un ejemplo):

Primero obtenemos el servicio para “inflar” layouts

```
LayoutInflater li = (LayoutInflater) contexto.getSystemService(Context.LAYOUT_INFLATER_SERVICE);  
View fila = li.inflate(R.layout.movimiento, null);
```



La variable **fila** contiene todo el layout inflado en memoria (es una vista). Ahora solo queda añadirlo donde sea conveniente dentro de nuestra vista



Después, inflamos el layout que queramos

# Diseño de GUI con Android Studio

- **Ejercicio 8.12**
- Vamos a crear una clase que implemente la lógica del juego
- Representaremos cada jugador con un carácter.
  - El jugador humano estará representado por la 'X'
  - La máquina por 'O'.
  - Las casillas en blanco con un carácter blanco.
- En cuanto a las posiciones en el panel de juego, estarán numeradas de 0 a 8 comenzando por la esquina superior izquierda y avanzando de izquierda a derecha y de arriba a abajo.
- Los métodos que se deben implementar son:
  - void limpiarPanel()
  - boolean mover(int posición, char jugador)
  - char compruebaGanador()
  - int movimientoDeLaMaquina()
  - void estableceDificultad(int dificultad)
  - char turno()
  - char [] estado()
- Los detalles de todos estos métodos los puedes encontrar en el PDF.

# Diseño de GUI con Android Studio

- **Ejercicio 8.12 (cont.)**
- Ahora vamos a diseñar la vista del “Tres en Raya”, es decir, dotar de dimensión gráfica a la lógica que se ha implementado anteriormente.
- Para ello vamos a proceder como sigue:
  - En primer lugar vamos a diseñar el layout principal de la aplicación.
  - Vamos a preparar un adaptador para el GridView.
  - A continuación interceptaremos el evento de pulsación de una celda.
  - Implementar el botón para iniciar una nueva partida.

# Internacionalización

- Hasta ahora hemos desarrollado nuestras aplicaciones en un solo idioma (español).
- Android permite adaptar fácilmente cualquier aplicación a otros idiomas y regiones.
- Para ello, el desarrollador debe colocar todos aquellos recursos con dependan de la región en la carpeta **res** y crear una versión de dichos recursos para cada idioma que quiera soportar.
- Gracias a que los recursos se encuentran “fuera” del código fuente, el desarrollador puede centrarse en el código de la aplicación mientras que un traductor (que no tiene que saber programación) puede centrarse en la localización de la aplicación a otro idioma/región.

# Internacionalización

- Existen distintos tipos de recursos en android que son susceptibles de localización: números enteros, cadenas, colores, layouts, animaciones, etc.
- Todos estos recursos deben ubicarse dentro de la carpeta **res**. Esta a su vez contiene carpetas específicas para cada tipo de recurso
- Las más comunes son:
  - res/values (contiene valores de tipos básicos: cadenas, enteros, etc.)
  - res/layout (contiene layouts)
  - res/animator (contiene animaciones)
  - res/drawable (contiene imágenes)
  - res/raw (contiene ficheros no procesados)
  - res/menu (contiene menús)

# Internacionalización

- ¿Cómo podemos acceder a estos recursos para usarlos en nuestra aplicación?
- Si queremos acceder a ellos desde el código, usamos la clase R.
- Las herramientas de construcción de Android generan esta clase explorando la carpeta res y crean un identificador para cada recurso, de forma que sea fácilmente accesible desde el código fuente.
- Esta forma de acceso la hemos utilizado bastante hasta:

```
// Set the text on a TextView object using a resource ID  
TextView msgTextView = (TextView) findViewById(R.id.msg);  
msgTextView.setText(R.string.hello_message);
```



En la clase R los identificadores de recursos están clasificados por tipo

# Internacionalización

- Si queremos acceder a los recursos desde los propios ficheros de recursos (ficheros XML) usamos la notación **@tipo/nombre**.

```
<?xml version="1.0" encoding="utf-8"?>
<EditText xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:textColor="@color/opaque_red"
    android:text="@string/hello" />
```

Este es un recurso de tipo string (cadena de caracteres) que se utilizará en el atributo “android:text” de este componente

Este es un recurso de tipo color

# Internacionalización

- ¿Cómo podemos localizar la aplicación a otro idioma?
- Lo único que tenemos que hacer es proporcionar los recursos de dicho idioma en una carpeta con el mismo nombre que la original pero con un sufijo que indique el idioma.
- Por ejemplo:
  - `res/values-es` (valores para español)
  - `res/values-fr` (valores para francés)
  - `res/values-ja` (valores para japonés)
  - `res/drawable-es` (imágenes para español)
  - `res/raw-de` (no procesados para alemán)
- Android usa por defecto el conjunto de recursos que se corresponde con la configuración regional del dispositivo.
- **Si no encuentra tales recursos** (porque normalmente no ofrecemos la aplicación para todas las regiones e idiomas del mundo) **usará el conjunto de recursos que hay en las carpetas sin sufijo (recursos por defecto)**.

# Internacionalización

- A partir de ahora, todas las cadenas de caracteres (mensajes para el usuario, texto en etiquetas, en hints de EditText, etc.) irá en el fichero **strings.xml**, que contiene recursos de tipo cadena.
- Si queremos proporcionar una versión en otro idioma de la aplicación, solo tenemos que copiar dicho fichero, traducir las cadenas al nuevo idioma y ubicar el fichero en la carpeta **values-xx** correspondiente al nuevo idioma.
- Cuando queramos usar el texto, siempre haremos referencia al recurso con **R.string.nombre** o **@string/nombre** (dependiendo de dónde lo necesitemos).

# Internacionalización

- La mayoría de las clases de la GUI en Android poseen métodos para establecer el valor de una propiedad a partir de un identificador de recurso:

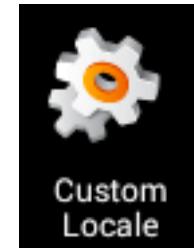
```
textoGanador.setText(R.string.mastermind_text);
```

- Pero hay ocasiones en las que necesitamos almacenar el recurso en una variable para manipularlo.
- Siempre podemos acceder a los recursos a partir de un **contexto** recuperan un objeto **Resources** con el método **getResources()**:

```
String texto = getResources().getString(R.string.mastermind_text);
```

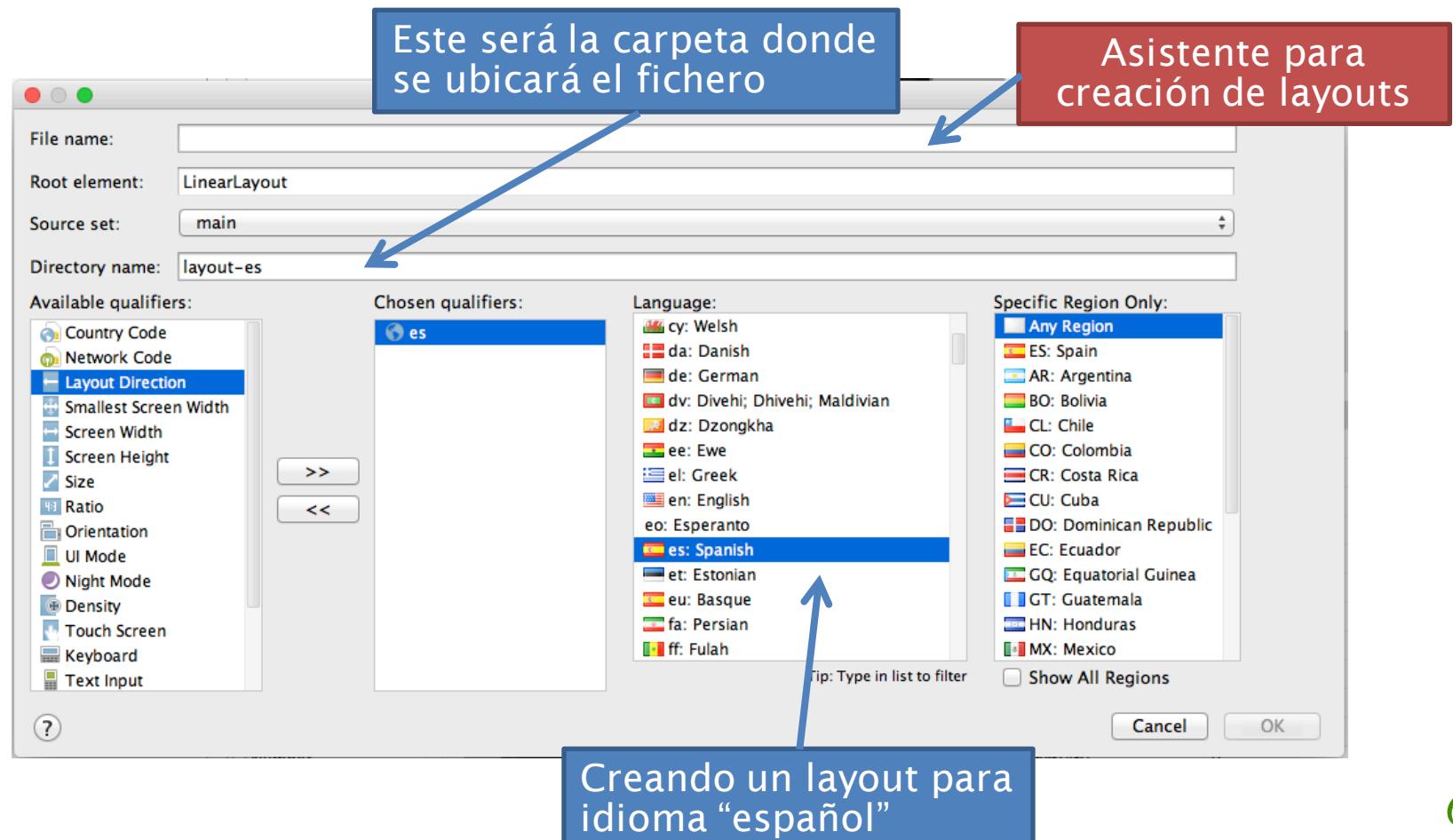
# Internacionalización

- ¿Cómo podemos probar si la aplicación traducida funciona correctamente?
- Podemos cambiar la configuración regional y de idioma en el dispositivo emulado gracias a la aplicación “**Custom Locale**”
- Una vez que seleccionemos el nuevo idioma, nuestra aplicación automáticamente debería mostrarse en el nuevo idioma.
- En un dispositivo real también podemos cambiar la configuración regional y el idioma para probar la aplicación.



# Internacionalización

- Android Studio ofrece asistentes para ayudarnos en la generación de recursos localizados



# Internacionalización

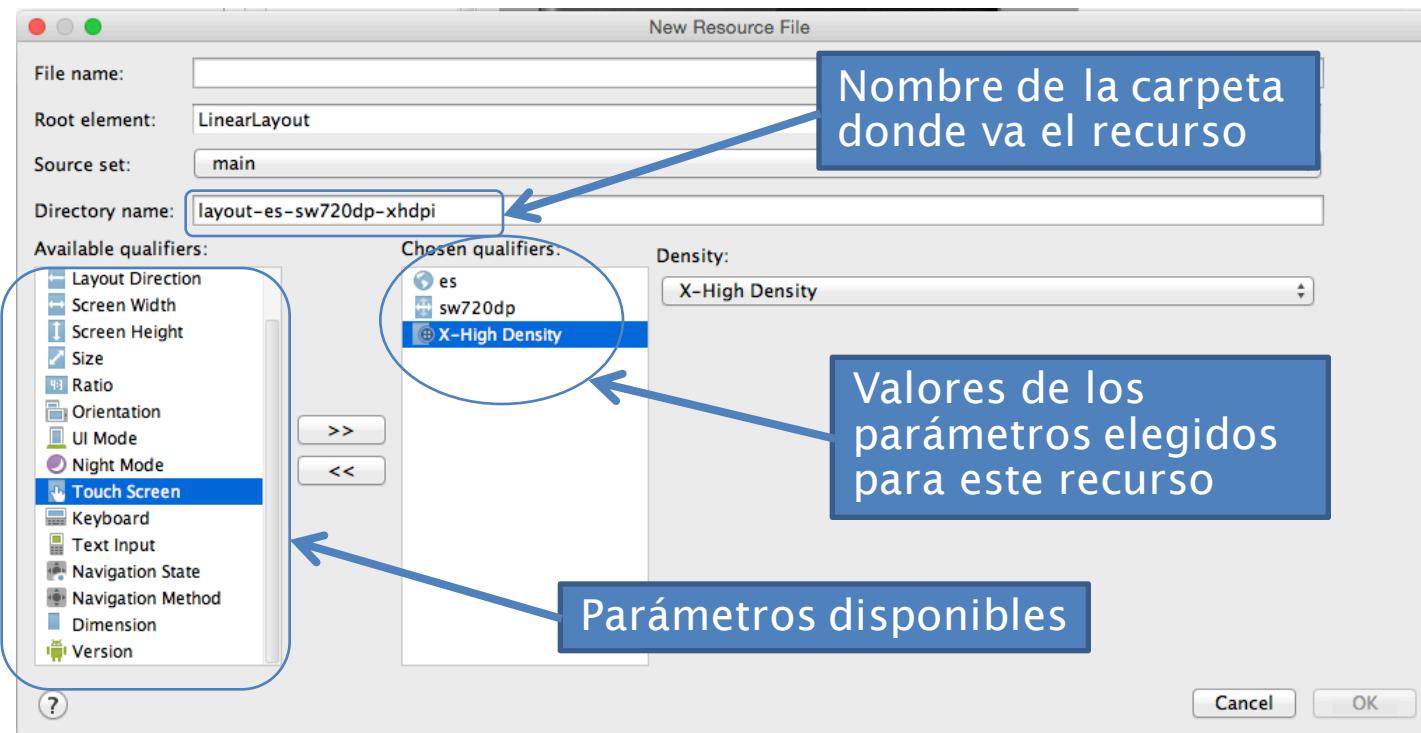
- **Ejercicio 8.13**
- Vamos a localizar el Tic-Tac-Toe del ejercicio 8.12 al inglés.
- Para ello vamos a traducir las cadenas que se muestran al usuario:
  - Empate -> **Draw**
  - ¡Has ganado! -> **You win!**
  - Lo siento, has perdido -> **Sorry, you lose**
  - Nuevo juego -> **New game**
  - Si necesitas traducir otra cadena, pregúntale al profesor

# Internacionalización

- El idioma no es el único parámetro que nos permite Android utilizar para determinar los recursos a utilizar.
- Es posible crear recursos específicos para distintas orientaciones del dispositivos, o distintos tamaños de pantalla, resoluciones, región, versión de Android, etc.
- Además todos estos parámetros se pueden combinar.
  - Por ejemplo, podemos proporcionar un layout que se use solamente cuando el dispositivo está apaisado, el idioma es francés y la resolución de pantalla es al menos 320 dpi.

# Internacionalización

- No conviene utilizar muchos parámetros simultáneamente, ya que el número de combinaciones crece exponencialmente.



# Internacionalización

- **Ejercicio 8.14**
- Vamos a modificar el Tic-Tac-Toe del ejercicio 8.13 para que el layout que parece cuando el dispositivo esté apaisado sea diferente al del dispositivo con orientación vertical.
- El layout para orientación vertical será el utilizado hasta ahora.
- El layout con orientación horizontal, mostrará los botones y el texto final a la derecha del tablero.
- **Ayuda:** el parámetro de orientación se llama “orientation” y sus valores pueden ser “land” (orientación horizontal) o “port” (orientación vertical).