Capítulo 3
Un poco más sobre UIs

Práctica 3.1: RelativeLayout

En la práctica 3.14 analizamos el LinearLayout, un contenedor que organiza los controles que se le añaden en una columna o en una fila. Es posible crear interfaces gráficos complejos anidando múltiples LinearLayout. Sin embargo, en ocasiones fuerza a jerarquías complicadas que requieren la creación de muchos objetos.

El RelativeLayout es un contenedor que añade a los controles que hospeda atributos adicionales para definir sus posiciones de una manera relativa a otros controles hermanos, o al padre (el propio RelativeLayout). El número de nuevos atributos es desbordante al principio, y no todos pueden ser utilizados simultáneamente.

Debemos tener cuidado cuando se utiliza este layout para evitar crear por error referencias circulares que impidan la colocación de los componentes. Además, en este caso suele ser más sencillo crear el layout directamente en XML, porque la herramienta gráfica de diseño es bastante difícil de domar.

La mayoría de los atributos que añade el RelativeLayout a sus controles hijo tienen como parámetro el identificador de otro control hermano. En ocasiones, sin embargo, son atributos booleanos. Para hacernos una idea clara de los atributos de este layout, es interesante organizarlos en función de qué lado del control sitúan. Es necesario tener en cuenta que esos atributos sólo colocan un lado. Por ejemplo, el atributo below[1] coloca el control que estamos especificando debajo de otro. Eso significa que el lado superior de dicho control estará en la misma horizontal que el lado inferior del control al que referenciamos. Pero no significa que estén exactamente debajo (tocándose). La posición en horizontal (izquierda-derecha) se establece con atributos diferentes.

- Colocación del lado superior:
  - ✓ **below**: coloca el control debajo de otro.
  - ✓ **alignTop**: coloca el lado superior en la misma horizontal que el lado superior de otro control.
  - ✓ **alignParentTop**: este atributo es booleano. Coloca el lado superior alineado con la parte superior del padre (el layout).
- Colocación del lado inferior:
  - ✓ **above**: coloca el control encima de otro.

[1] En realidad el nombre que hay que usar en el XML para el atributo es layout_below, pues es un parámetro para el layout (en el editor gráfico de Eclipse aparece en su apartado específico). Esto mismo ocurre con los demás atributos que mencionamos en el resto de la sección: deben ponerse siempre en el XML con el prefijo layout_ delante.

- ✓ **alignBottom**: coloca el lado inferior en la misma horizontal que el lado inferior de otro control.
- ✓ **alignParentBottom**: atributo booleano para colocar el lado inferior pegado a la parte inferior del padre (layout).

Colocación del lado izquierdo:

- ✓ **toRightOf**: coloca el control a la derecha de otro.
- ✓ **alignLeft**: coloca el lado izquierdo en la misma vertical que el lado izquierdo de otro control.
- ✓ **alignParentLeft**: atributo booleano para colocar el lado izquierdo pegado a la parte izquierda del padre (layout).

Colocación del lado derecho:

- ✓ **toLeftOf**: coloca el control a la izquierda de otro.
- ✓ **alignRight**: coloca el lado derecho en la misma vertical que el lado derecho de otro control.
- ✓ **alignParentRight**: atributo booleano para colocar el lado derecho pegado a la parte derecha del padre (layout).

Hay otros atributos que no encajan directamente en ninguna de esas categorías:

- **centerHorizontal**: atributo booleano que, de ser cierto, hace que el control quede centrado horizontalmente en el espacio del padre (layout).
- **centerVertical**: atributo booleano que, de ser cierto, hace que el control quede centrado verticalmente en el espacio del padre (layout).
- **centerInParent**: atributo booleano que fusiona los dos anteriores.
- **alignBaseLine**: alinea la _líneas base_ del control con el de otro. Se entiende por línea base al texto que contenga. Sirve por ejemplo para poner seguidos una etiqueta y un botón que, en principio, tendrán altos diferentes, pero que queremos que los textos estén alineados.
- **alignWithParentIfMissing**: valor booleano para determinar que si el control al que referenciamos en alguno de los atributos no está (su visibilidad es gone), entonces debemos utilizar como referencia al padre (el layout).

Debe tenerse en cuenta que no hace falta "fijar" todos los lados de los controles, dado que aún se hará uso de los atributos layout_width y layout_height. Además, como se ha dicho, no se debe hacer un uso contradictorio de los atributos (por ejemplo, no tiene sentido establecer al mismo tiempo **toLeftOf** y **centerHorizontal**).

Para dar más posibilidades de colocación, recuerda que puedes también modificar los atributos de margen de los controles, con lo que se consigue primero alinear un control con respecto a otro, y luego _desplazarlo_ gracias al margen.

Por último, es importante evitar las referencias circulares, pero esto se refiere a referencias de lados iguales. Sí podremos, sin embargo, establecer por ejemplo el lado izquierdo del control A en relación con el control B, y a su vez el lado superior del B en relación con el del A. Cuando se hacen construcciones de ese tipo, en el XML ¿qué

control definimos antes? ¿El A o el B? Dado que ambos referencian al otro, tendremos un potencial problema por identificadores desconocidos. Si eso ocurre, se puede referenciar a un control que aún no existe poniendo el + como en la creación de los controles.

A modo de ejemplo, para definir el layout de la figura:

usaríamos el código:

```
<TextView
        android:id="@+id/tvNombre"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/etNombre"
        android:padding="8dp"
        android:text="@string/nombre"/>
<EditText
        android:id="@+id/etNombre"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_toRightOf="@+id/tvNombre"
        android:layout_alignParentTop="true"/>
```

¿Eres capaz de conseguir, usando un RelativeLayout, la disposición de la práctica 2.15?

Práctica 3.2: Botones de radio

Los botones de radio (**RadioButton**) son controles en los que se da al usuario la posibilidad de seleccionar una de varias opciones, y el hecho de seleccionar una hace que las demás se desmarquen. Los botones de radio se agrupan en **RadioGroup**, que se encarga de garantizar que sólo uno de los botones de radio esté activo en un determinado momento.

La clase **RadioGroup** hereda de LinearLayout, por lo que podremos orientarlo tanto horizontal como verticalmente. Se espera que en el interior de un RadioGroup sólo haya objetos de la clase RadioButton, aunque no es obligatorio.

Por su parte, el atributo más destacado de los *RadioButton* es ***checked***, que nos indica si está o no seleccionado.

Para saber en código qué opción está seleccionada, tenemos dos opciones. La primera es recorrer todos los RadioButton y mirar cuál tiene el atributo checked a cierto. La segunda, mucho más práctica, es preguntarle al RadioGroup el identificador del RadioButton seleccionado (método getCheckedRadioButtonId()).

- Crea un nuevo proyecto de nombre RadioGroup con los valores habituales.
- Crea un interfaz como el de la figura.

Coloca una etiqueta oculta en la parte inferior de la ventana.

- Añade código al evento del botón para que escriba en la etiqueta inferior un comentario en relación con la selección (por ejemplo "Vaya con la manzanita..."). Para eso, puedes usar el método del RadioGroup **getCheckedRadioButtonId**() (que tendrás que haber conseguido con

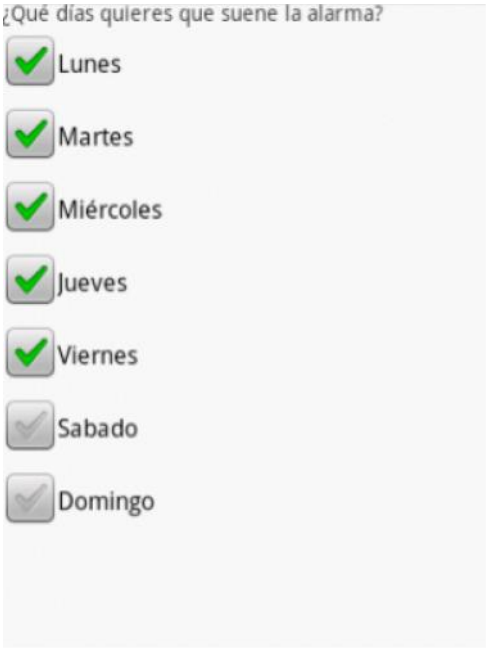
findViewById()), y luego hacer un **switch** para compararlo con cada uno de los identificadores de los RadioButton.

- También puedes ahorrarte el botón de aceptar, y mostar el mensaje cada vez que se modifica la selección. Para eso, puedes registrarte como oyente del RadioGroup usando un código como:

```
RadioGroup rg = ...;
rg.setOnCheckedChangeListener(
        new RadioGroup.OnCheckedChangeListener() {
                public void onCheckedChanged(RadioGroup group,
                                                int checkedId) {
                        // Hacer algo...
                }
        }
);
```

Práctica 3.3: Casillas de verificación

Una casilla de verificación es un control tipo botón especial que tiene dos posibles estados, marcado y desmarcado. Muestra un texto y una parte que representa el estado. Para desarrollar su comportamiento, la clase CheckBox hereda de Button.



Tiene los atributos **android:text y android:checked** que podríamos esperar. Para saber desde código si está o no marcado, podemos usar boolean **isChecked**(). Por último, para detectar los cambios de estado, tendremos que registrarnos como observadores suyos:

```
CheckBox cb = ...
cb.setOnCheckedChangeListener(
```

```
        new CompoundButton.OnCheckedChangeListener() {
                public void onCheckedChanged(CompoundButton buttonView,
                                                        boolean isChecked) {
                        // Hacer algo! Sabemos quién ha sido pulsado
                        // y su estado
                } // onCheckedChanged
        } // new
); // setOnCheckedChangeListener
```

También podemos, sencillamente, capturar el evento **onClick** como con los botones. En ese caso, usaremos **isChecked**() para saber si está o no seleccionado. Dado que el evento **onClick** recibe la vista que lo ha generado, puedes convertir el parámetro a la clase **CheckBox** y llamar al método sin problema.

- Crea un proyecto nuevo llamado CheckBox con las opciones habituales.
- Define la disposición de la ventana como la de la figura.
- En la parte inferior, coloca una etiqueta invisible con el texto ¡No! ¡El domingo no!, que se haga visible cuando se seleccione el domingo.

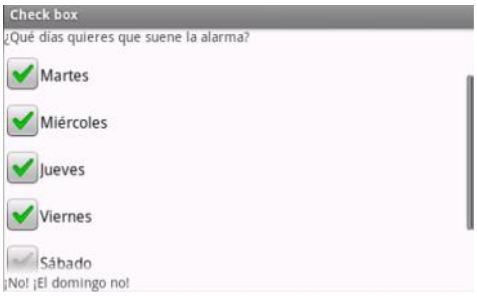
Práctica 3.4: ScrollView

En la práctica anterior, si se coloca el dispositivo en apaisado (landscape), las casillas de verificación no entran completamente, y no puede accederse a ellas. Recuerda que para rotar el dispositivo en el AVD puedes usar Ctrl-F11 o Ctrl-F12 (o las teclas de los números 7 y 9 del teclado numérico sin el bloqueo numérico activado).

Android nos proporciona un tipo especial de layout llamado **ScrollView** que sirve para hospedar un único componente (que normalmente será otro layout) y que permite desplazarse verticalmente en él si ocupa más espacio que el disponible en la ventana.

Sabiendo esto, modifica la práctica para que se pueda utilizar también con el móvil en posición apaisada.

¿Eres capaz de hacerlo de manera que las etiquetas "floten" y el ScrollView sea únicamente para los cuadros de verificación?



Práctica 3.5: Mensajes rápidos: toasts

Para proporcionar al usuario información rápida, de poca importancia y de manera poco intrusiva Android proporciona las que denomina toasts ("tostadas"). El nombre hace referencia a las tostadas que "saltan" en un tostador para avisar de que ya están listas, y luego vuelven hacia dentro y desaparecen. En Android, son pequeños textos que se muestran brevemente al usuario "flotando" sobre la ventana principal, y desaparecen automáticamente un instante después.

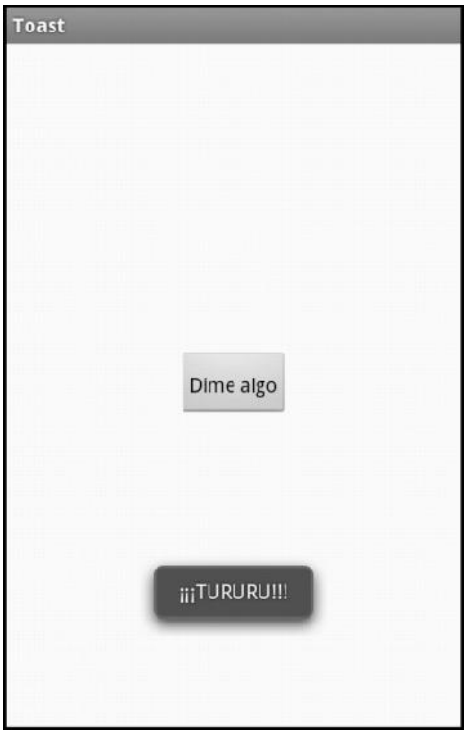
El esquema de uso es sencillo:

```
Context contexto = getApplicationContext();
int duracion = Toast.LENGTH_SHORT; // Toast.LENGTH_LONG

Toast toast = Toast.makeText(contexto,
                        R.string.<recurso>,
                        duracion);
toast.show();
```

En este código se ha asumido que la cadena la tenemos como recurso (recomendado). También hay un método que recibe una cadena.

- Crea un proyecto nuevo que muestre un botón que al ser pulsado muestre un mensaje usando un Toast.



El método **Toast.makeText()** devuelve una instancia de la clase Toast con una con_guración predefinida. Es posible crear instancias manualmente y personalizar el layout que mostrarán, en cuyo caso aprovecharemos la clase Toast para que se encargue de hacer visible nuestro aviso durante un breve espacio de tiempo.

Práctica 3.6: Mensajes de alerta con botones

Android permite la creación de cuadros de diálogo a través de su clase **Dialog**. Además, para hacer cuadros de diálogo sencillos (por ejemplo, para pedir confirmación de una acción), existe la clase **AlertDialog** que nos simplifica la tarea. Incluso podemos utilizar una clase auxiliar, llamada **AlertDialog.Builder**, que nos facilita la con_guración del cuadro de diálogo de alerta.

Por ejemplo, para mostrar un cuadro de diálogo con un único botón, podemos usar el siguiente esquema:

```
AlertDialog.Builder alert = new AlertDialog.Builder(this);
alert.setMessage(R.string.<id>);
alert.setPositiveButton(android.R.string.ok, null);
alert.show();
```

Algunos comentarios:

- Para establecer el texto a mostrar usamos **setMessage**(), pasando el identificador del recurso o una cadena cableada.
- Podemos establecer el título del cuadro de diálogo con **setTitle**(), pasando también el identificador de una cadena o la cadena directamente. Ten en

cuenta que si el mensaje es corto, el resultado es poco claro, porque el título y el mensaje podrían salir con el mismo tamaño de letra.

- El método setPositiveButton() tiene dos parámetros.
  - ✓ El primero indica la cadena a mostrar. En el ejemplo, observa que hemos utilizado un recurso predefinido de Android, en concreto android.R.string.ok, que contiene el texto (localizado al idioma del sistema) de "Ok" (aceptar). Esto nos ahorra trabajo de localización a nosotros, aunque puede hacer que un usuario vea toda la aplicación en español, salvo los botones de los diálogos que le saldrán en su idioma nativo.
  - ✓ El segundo recibe una instancia de una clase interna pública, **DialogInterface.OnClickListener** a la que se llamará cuando se pulse el botón. En este caso pasamos **null** porque no nos interesa la recepción del mensaje.
- Podemos usar **setNegativeButton**(), que tiene el mismo prototipo que **setPositiveButton**(), pero ahora está pensado para el botón de "no".
- **setNeutralButton**() es similar al anterior, pero para el botón neutral (por ejemplo para el tercer botón de "Sí", "No", "Cancelar").
- Podemos usar setIcon(<id>) para establecer un icono. Podemos aprovechar los iconos predefinidos de Android.

Vamos a probarlo.

- Haz un proyecto nuevo **AlertDialog** con un botón, que muestre un cuadro de diálogo al ser pulsado que tenga únicamente un botón de aceptar, y pruébalo.
- Haz que se muestre un Toast justo después de lanzar el cuadro de diálogo.

El resultado de la última ejecución sorprende. Lo más destacado de los cuadros de diálogo en Android es que no son modales. Es decir, cuando se muestra uno, no se congela la ejecución. Por eso habrás visto aparecer el toast antes de cerrar el diálogo, puesto que la aplicación sigue ejecutándose (aunque afortunadamente la entrada se bloquea para toda la actividad).

Esto significa que la construcción tradicional en Windows de:

```
DialogResult resultado = MessageBox(this, mensaje,
                                    titulo, botones);
if (result == DialogResult.Yes) {
...
}
```

no puede hacerse en Android. Los cuadros de diálogo no devuelven un valor. Para crear un funcionamiento de este tipo, tendrás que hacerlo de manera asíncrona, metiendo en tu actividad un evento que se llame cuando el usuario acepte en el cuadro de diálogo.