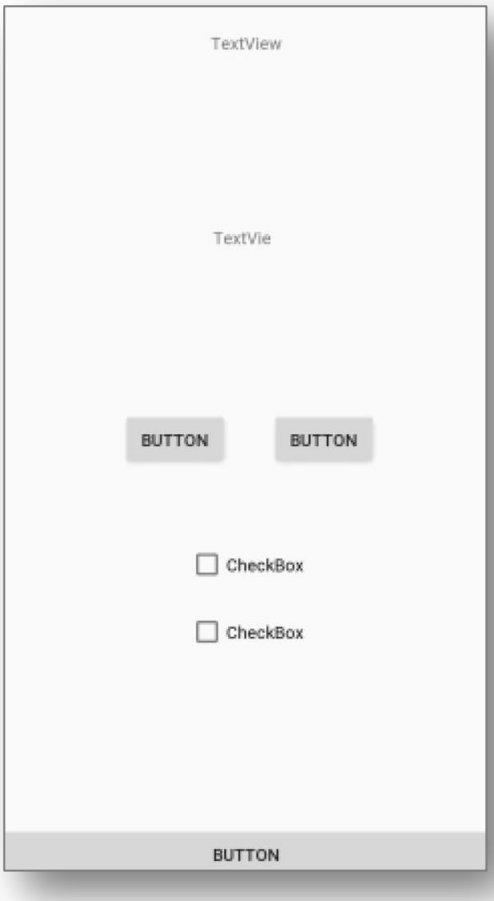
El Contador

En esta práctica vamos a volver a usar los conocimientos adquiridos en la práctica anterior y vamos a ver algunas cosas más, como hacer que nuestra app se pueda ejecutar en vertical y en horizontal, para ello veremos cómo definir nuestro layout para ambos casos y como conseguir que no se pierdan los valores de los elementos de la interfaz cuando hacemos el giro del dispositivo (cambio de layout).

Fase 1

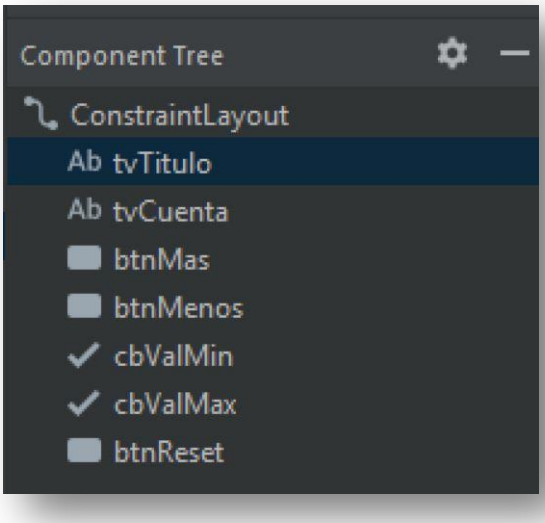
Como en la práctica anterior lo primero que vamos a hacer establecer cómo va a ser nuestra interfaz, que elementos tiene y como los distribuimos:



Como podemos apreciar nuestra interfaz consta de los siguientes elementos

- TextView → donde podemos poner un título, una información para el usuario, etc.
- TextView→ Aquí mostraremos la cuenta, empezaremos en 0.
- Button → Nos servirá para incrementar la cuenta
- Button → Nos permitirá decrementar la cuenta
- CheckBox → indicamos si permitimos valores negativos
- CheckBox → indicamos si permitimos valores superiores a 50
- Button → Resetea la cuenta a 0.

En esta primera fase tenemos que conseguir esta ubicación de los elementos y asignarles los siguientes identificadores:



FASE 2

Como en la práctica anterior, vamos a ir dándole nuestro toque personal a la app, cambiaremos los atributos de los elementos de la interfaz y vamos añadiendo la información para que el usuario sepa cómo funciona.

Queremos que quede como se ve en la imagen inferior:

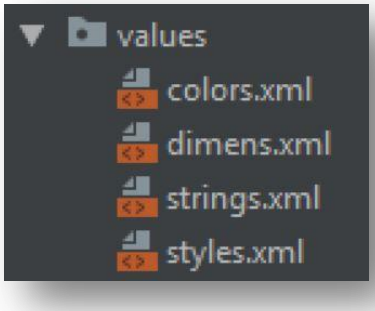
Como se puede ver hemos aplicado una fuente propia, hemos establecido varios estilos de texto:

- Base
- Titulo (lleva un cambio de color)
- Cuenta → el tamaño de la cifra es de 100dp (ojo no ds)

A los CheckBox les hemos asociado nuestra fontFamily y el tamaño de texto Base.

Lo mismo hemos hecho con los botones a los que además hemos cambiado el color de texto y en el caso del reset hemos hecho que no aparezcan las letras en mayúscula

Hay que ir acostumbrándose a trabajar con los ficheros de recursos:



El Contador

Fase 3

Ahora vamos a la parte de código java como ya tenemos asignados los id a los elementos del layout podemos usarlos en nuestra App.

Lo primero que vamos a hacer es cambiar el nombre a nuestra clase MainActivity y la llamamos Contador, usamos Refactor/Rename y le decimos que cambie el nombre en todas partes.

De igual forma vamos a cambiar el nombre del layout, le llamamos contador_lay.

En nuestra interfaz tenemos 3 botones, nos damos cuenta de que la forma de asociar los listeners y definir sus acciones en la práctica anterior conlleva repetir código y hacerlo más engorroso, esto hay que mejorarlo.

Para agilizar el uso de los botones, vamos a Importar el método de la clase View que nos permite definir qué hacer cuando pulsan alguno de los botones, veamos cómo queda la declaración de nuestra clase principal:

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

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

    @Override
    public void onClick(View view) {

    }//fin onClick
}//FIN Contador
```

Ahora podemos declarar nuestros botones y asociarlos todos a este método onClick, y en el definir que hacer según que botón haya sido pulsado por el usuario

Ahora vamos a declarar como variable privadas dentro de la clase Contador, los elementos del layout que vamos a necesitar controlar desde el código, como ya sabemos el tipo es el mismo que el elemento que hemos definido en la vista y las inicializamos conectándolas con los elementos de la vista mediante el id de estos, es decir, como hicimos en la práctica anterior.

Veamos como quedaría el código

```
private TextView cuenta;
private Button mas, menos, reset;
private CheckBox min, max;

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

    cuenta = findViewById(R.id.tvCuenta);
    //Conectamos con el layout y el listener los botones
    mas = findViewById(R.id.btnMas);
    mas.setOnClickListener(this);
    menos = findViewById(R.id.btnMenos);
    menos.setOnClickListener(this);
    reset = findViewById(R.id.btnReset);
    reset.setOnClickListener(this);
    //Conectamos con el layout y el listener los checkbox
    min = findViewById(R.id.cbValMin);
    min.setOnClickListener(this);
    max = findViewById(R.id.cbValMax);
    max.setOnClickListener(this);

    //Estamos usando el mismo listener para los botones
    //y para los checkbox, ya que todos son vistas y
    //por tanto heredan de la clase View.
}//fin onCreate
```

Cuando establecemos el listener hacemos referencia mediante THIS al método ONCLICK que nos hemos traído mediante el IMPLEMENTS y que a continuación vemos.

El Contador

La estructura del método onClick suele ser la siguiente:

```
@Override
public void onClick(View view) {
    switch(view.getId())
    {
        case R.id.btnMas:
            break;
        case R.id.btnMenos:
            break;
        case R.id.btnReset:
            break;
        case R.id.cbValMax:
            break;
        case R.id.cbValMin:
            break;
        default: break;
    }
}//fin onClick
```

Esa variable VIEW que nos llega como parámetro nos da la información del objeto que ha provocado la activación del listener, y por supuesto entre esa información se encuentra el id que le hemos asignado y que obtenemos mediante el método GETID().

Los id quedan codificados como números por tanto, podemos usarlos como posibles opciones dentro de un SWITCH.

Ahora lo que tenemos que hacer es acabar la codificación de lo que queremos que nuestra app haga en cada caso, para ello necesitaremos unas variables booleanas y estaría bien tener una variable entera que vaya almacenando la cuenta.

Las vamos a definir también como variables de la clase.

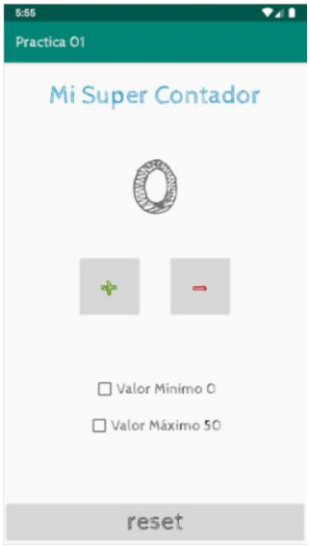
```
private boolean isVmin = false;
private boolean isVmax = false;
private final static int ValorMax = 50;
private final static int ValorMin = 0;
private int ValCuenta = 0;
```

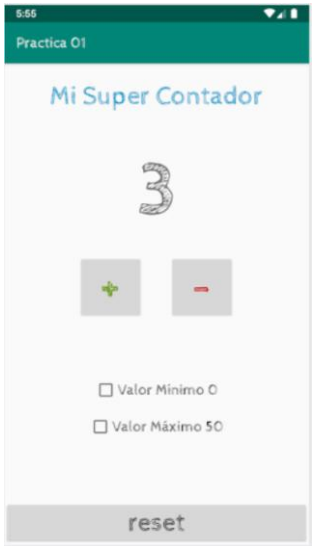
Completad el código del método onClick teniendo en cuenta todas las condiciones

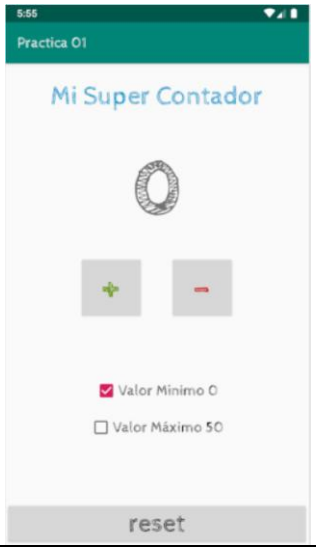
- PULSAR **+** → incrementamos la cuenta en 1 (OJO CON VALOR MAX)
- PULSAR **-** → decrementamos la cuenta en 1 (OJO CON VALOR MIN)
- PULSAR **reset** → ponemos la cuenta a 0
- MARCAR VALOR MIN → No puede haber nº negativo, si la cuenta es negativa en ese momento la ponemos a 0.
- MARCAR VALOR MAX → No puede haber nº mayor que 50, si la cuenta es mayor en ese momento la ponemos a 50.

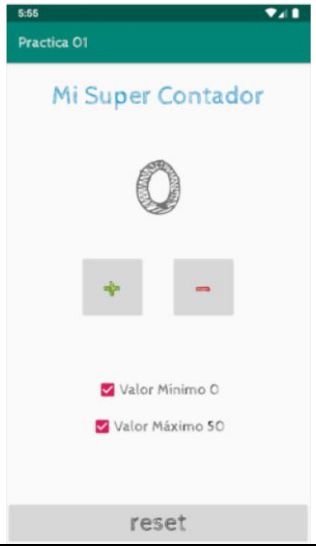
Veamos a continuación una ejecución de prueba

PANTALLA INICIAL	PULSAMOS 3 VECES +	PULSAMOS 5 VECES -
MARCAMOS VALOR MÍNIMO	LLEVAMOS CUENTA A 60	MARCAMOS VALOR MÁXIMO
PULSAMOS RESET		







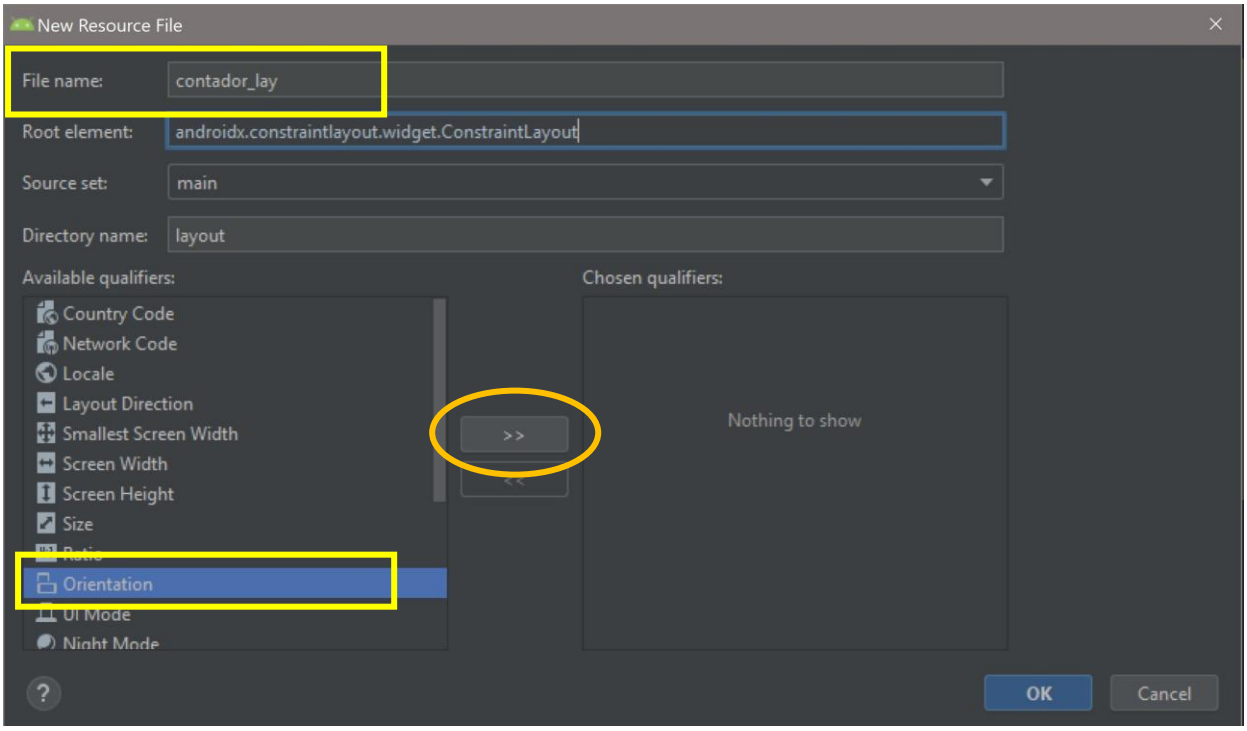


El Contador

Fase 4

Ahora queda el detalle final, hacer nuestro layout horizontal (landscape), es decir, hacemos otro layout que tiene que quedar así:

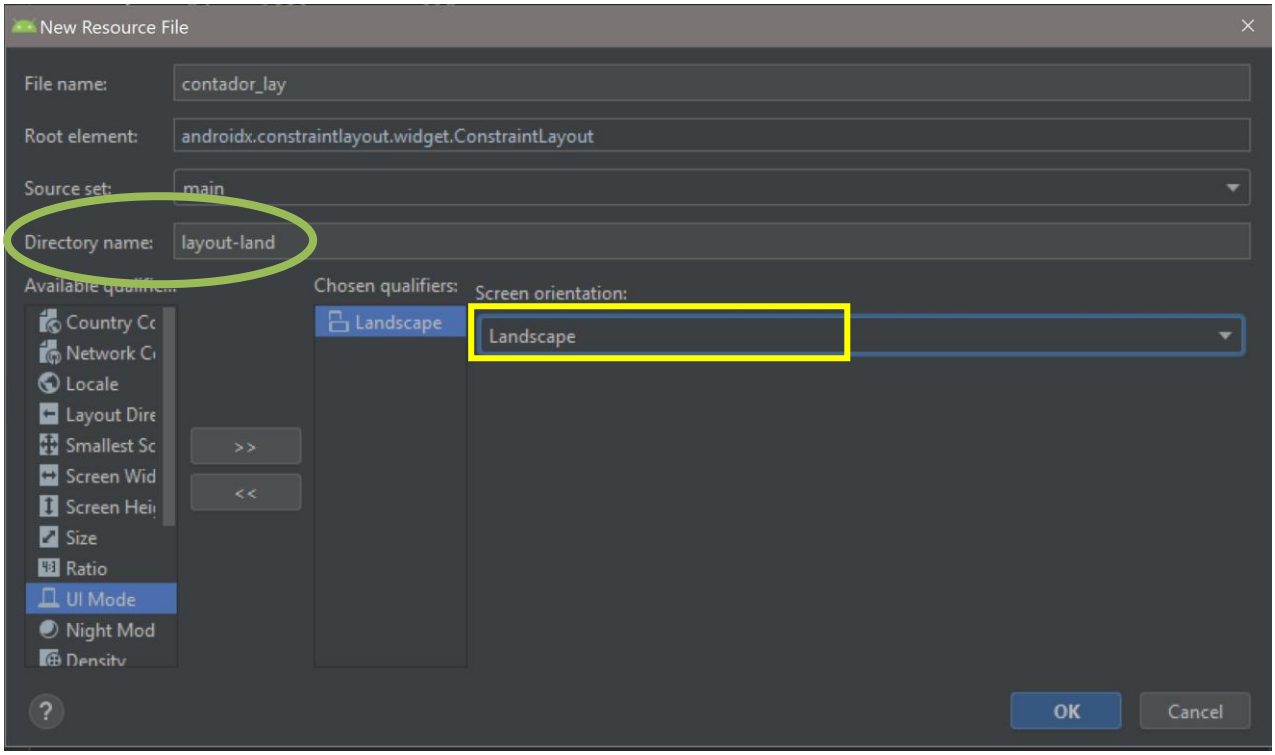
Lo que ocurrirá es que empezaremos trabajando en vertical (portrait) y cuando giremos el móvil entrará a funcionar el layout landscape, para ello se tienen que cumplir unas características a la hora de crear este nuevo fichero:



Le ponemos exactamente el mismo nombre que el layout que ya tenemos en vertical.

En la parte de abajo, en cualificadores, elegimos Orientation y pulsamos en el botón de la doble flecha a la derecha.

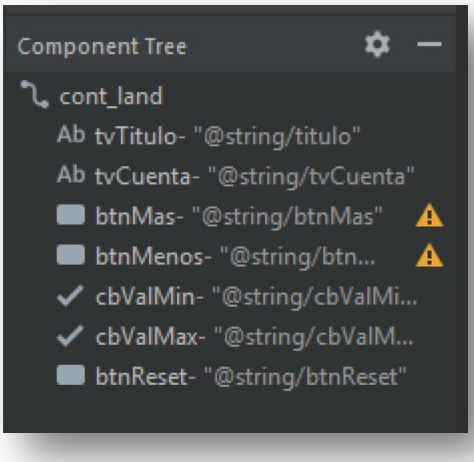
Así llegaremos a la siguiente pantalla:



En el desplegable Screen orientation seleccionamos Landscape, en ese momento nos fijamos en cómo cambia el nombre del directorio con respecto a la pantalla anterior.

Lo que hace Android es crear una carpeta dentro de los layouts donde va a guardar las versiones landscape de los layouts.

Otra característica a tener en cuenta, fijaros en los identificadores de los elementos que tenemos en el layout horizontal:



No, no me he equivocado, fijaos que le he puesto un id distinto al layout.

Usamos los mismos identificadores para los elementos del layout.

De esta forma nuestro código JAVA no se ve afectado en absoluto, la app funciona exactamente igual en vertical que en horizontal.

Pero para que esto funcione correctamente debemos añadir una característica a nuestra Activity en el Android Manifest:

```
<activity android:name=".Contador"
          android:screenOrientation="fullSensor">
```

(MIRAR EL ANEXO DE LA PRÁCTICA)

El Contador

ANEXO I SCREENORIENTATION

Define la orientación en la que se mostrará la actividad en el dispositivo.

El sistema ignora este atributo si la actividad se ejecuta en modo de ventanas múltiples.

El valor puede ser una de las siguientes strings:

`"unspecified"`	Valor predeterminado. El sistema selecciona la orientación. La política que usa (y en consecuencia, las elecciones que realiza en contextos específicos) puede variar de un dispositivo a otro.
`"behind"`	La misma orientación de la actividad inmediatamente debajo de ella en la pila de actividades.
`"landscape"`	Orientación horizontal (la pantalla es más ancha que alta).
`"portrait"`	Orientación vertical (la pantalla es más alta que ancha).
`"reverseLandscape"`	Orientación horizontal en la dirección opuesta al modo horizontal convencional. *Se agregó en nivel de API 9.*
`"reversePortrait"`	Orientación vertical en la dirección opuesta al modo vertical convencional. *Se agregó en nivel de API 9.*
`"sensorLandscape"`	Orientación horizontal, pero puede ser modo horizontal convencional o inverso según el sensor del dispositivo. *Se agregó en nivel de API 9.*
`"sensorPortrait"`	Orientación vertical, pero puede ser modo vertical convencional o inverso según el sensor del dispositivo. *Se agregó en nivel de API 9.*
`"userLandscape"`	Orientación horizontal, pero puede ser modo horizontal convencional o inverso según el sensor del dispositivo y la preferencia del usuario con respecto al sensor. Si el usuario bloqueó la rotación definida por sensor, este valor funciona igual que `landscape`. De lo contrario, se comporta igual que `sensorLandscape`. *Se agregó en el nivel de API 18.*
`"userPortrait"`	Orientación vertical, pero puede ser modo vertical convencional o inverso según el sensor del dispositivo y la preferencia del usuario con respecto al sensor. Si el usuario bloqueó la rotación definida por sensor, este valor funciona igual que `portrait`. De lo contrario, se comporta igual que `sensorPortrait`. *Se agregó en el nivel de API 18.*
`"sensor"`	El sensor de orientación del dispositivo determina la orientación. La orientación de la pantalla depende de cómo el usuario sostiene el

	dispositivo. Cambia cuando el usuario gira el dispositivo. Sin embargo, algunos dispositivos no rotarán en las cuatro orientaciones posibles de forma predeterminada. Para permitir la rotación en las cuatro orientaciones, usa `"fullSensor"`.
`"fullSensor"`	El sensor de orientación del dispositivo determina la orientación para cualquiera de las 4 orientaciones. Es similar a `"sensor"`, con la excepción de que aquí se permite cualquiera de las cuatro posibles orientaciones de pantalla, independientemente del comportamiento normal del dispositivo (por ejemplo, algunos dispositivos no usan normalmente los modos vertical inverso u horizontal inverso, pero este valor los habilita). *Se agregó en el nivel de API 9.*
`"nosensor"`	La orientación se determina sin hacer referencia a un sensor de orientación físico. Se ignora el sensor; en consecuencia, la pantalla rotará independientemente del movimiento del usuario. Salvo por esta diferencia, el sistema elige la orientación con la misma política de la configuración "`unspecified`".
`"user"`	La orientación que prefiere el usuario actualmente.
`"fullUser"`	Si el usuario bloquea la rotación definida por sensor, este valor funciona igual que `user`. De lo contrario, se comporta igual que `fullSensor` y admite cualquiera de las cuatro posibles orientaciones de pantalla. *Se agregó en el nivel de API 18.*
`"locked"`	Bloquea la orientación en la rotación actual, sin importar la que sea. *Se agregó en nivel de API 18.*

El Contador

ANEXO II SAVEDINSTANCESTATE

Cuando se producen cambios en el estado de la APP, por ejemplo cuando giramos el móvil, los valores que no estén guardados desaparecen, para evitarlo podemos hacer los siguiente:

```
//Vamos a guardar el estado de ValCuenta
private final static String CUENTA = "cuenta_actual";
private int ValCuenta;
```

```
//Sobreescribimos los métodos para guardar y recuperar el estado de la instancia
        @Override
        public void onSaveInstanceState(Bundle savedInstanceState) {
            super.onSaveInstanceState(savedInstanceState);

            savedInstanceState.putInt(CUENTA, ValCuenta);
        }

        @Override
        public void onRestoreInstanceState(Bundle savedInstanceState) {
            super.onRestoreInstanceState(savedInstanceState);

            if(savedInstanceState != null) {
                ValCuenta = savedInstanceState.getInt(CUENTA, defaultValue: 0);
            }
            else ValCuenta = 0;
            cuenta.setText(""+ValCuenta);
        }
```