

Capítulo 2

Actividades: las ventanas de Android

Resumen: En este capítulo crearemos nuestras primeras aplicaciones para Android, explorando las llamadas "actividades" (activity), su ciclo de vida y sus interfaces gráficas.

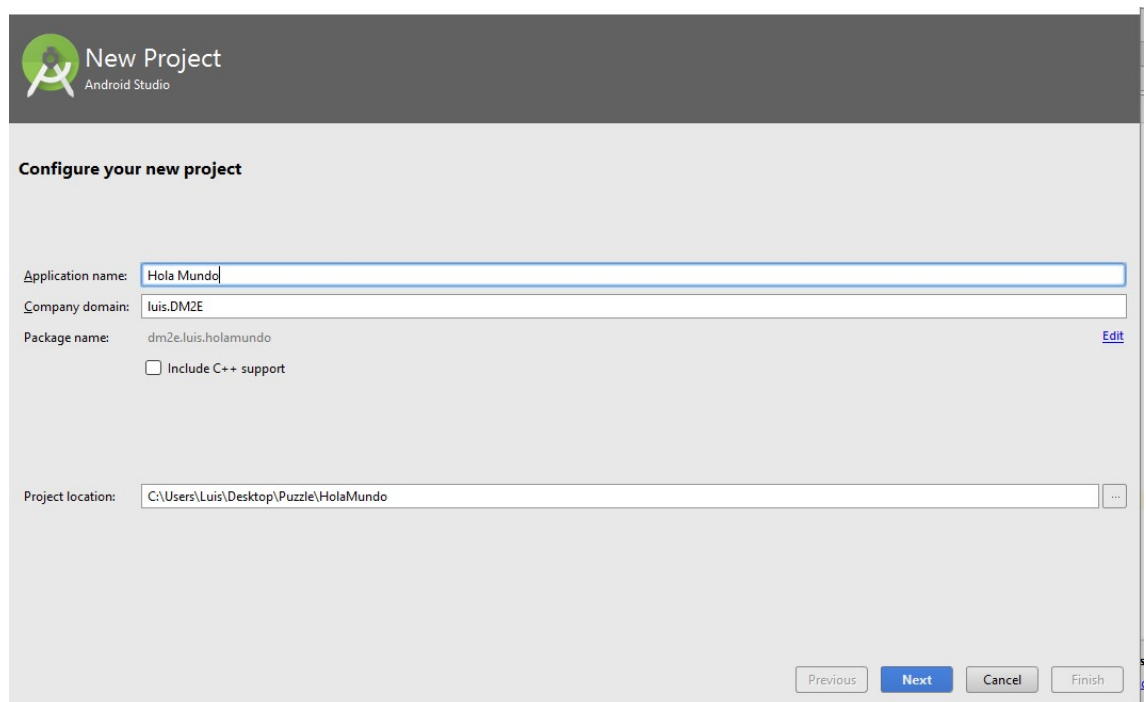
Práctica 2.1: Hola mundo

Vamos a hacer nuestro primer programa para Android, usando Android Studio. Lanza, si no lo has hecho ya, Android Studio.

Para lanzar el asistente de creación de un nuevo proyecto, ve a File/New /New Project.

La primera ventana del asistente nos pregunta por la información básica del proyecto:

- **Application name:** nombre de la aplicación que verá el usuario en el dispositivo. Especificaremos Hola mundo. Al escribirlo, Android Studio proporciona valores predefinidos para algunos de los demás campos.
- **Company Domain:** Dominio
- **Package name:** No hay que escribir utiliza el dominio y el nombre de la aplicación.
- Activador del soporte de C++.
- **Project Location:** ruta donde guardaremos nuestro proyecto.



The screenshot shows the 'New Project' dialog in Android Studio. The title bar says 'New Project' and 'Android Studio'. The main heading is 'Configure your new project'. Below this, there are four input fields: 'Application name' (containing 'Hola Mundo'), 'Company domain' (containing 'luis.DM2E'), 'Package name' (containing 'dm2e.luis.holamundo'), and 'Project location' (containing 'C:\Users\Luis\Desktop\Puzzle\HolaMundo'). There is an unchecked checkbox labeled 'Include C++ support'. At the bottom right, there are four buttons: 'Previous', 'Next', 'Cancel', and 'Finish'.

- Pulsamos el botón Next.
- **Minimum SDK:** versión de la plataforma mínima que requiere la aplicación para poder ser ejecutada. Debe ser la menor posible para tu aplicación. Escoge API 14 o deja la 15.

☒ Phone and Tablet

Minimum SDK API 14: Android 4.0 (IceCreamSandwich)

Lower API levels target more devices, but have fewer features available.
By targeting API 14 and later, your app will run on approximately **100,0%** of the devices that are active on the Google Play Store.
[Help me choose](#)

☐ Wear

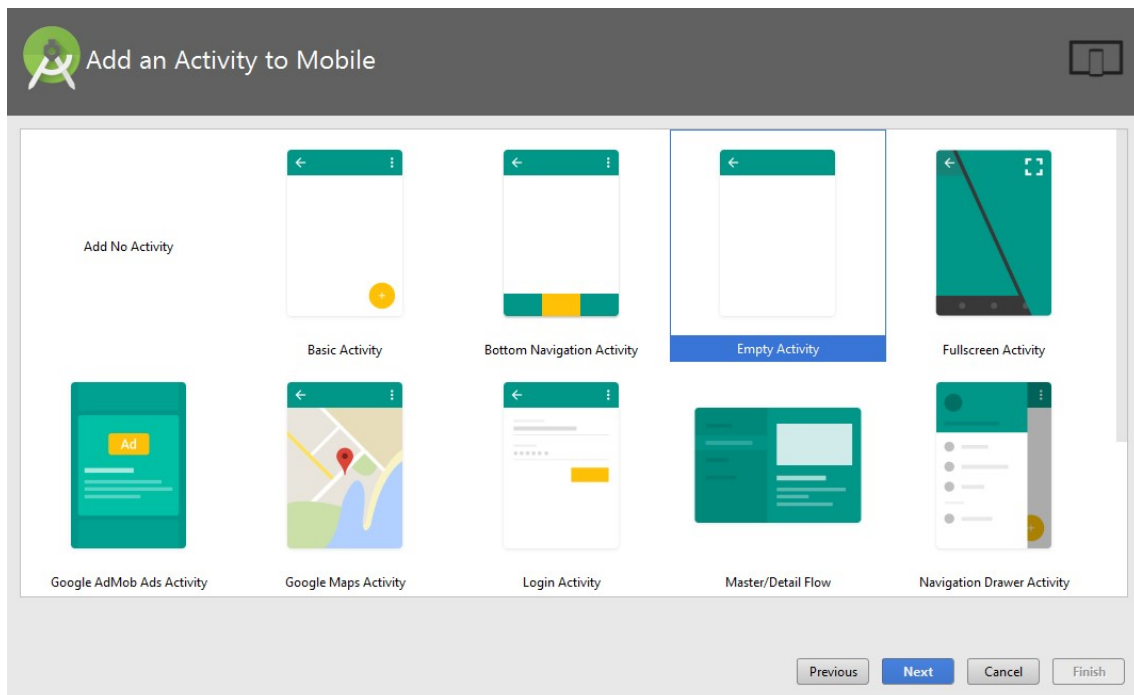
Minimum SDK API 21: Android 5.0 (Lollipop)

☐ TV

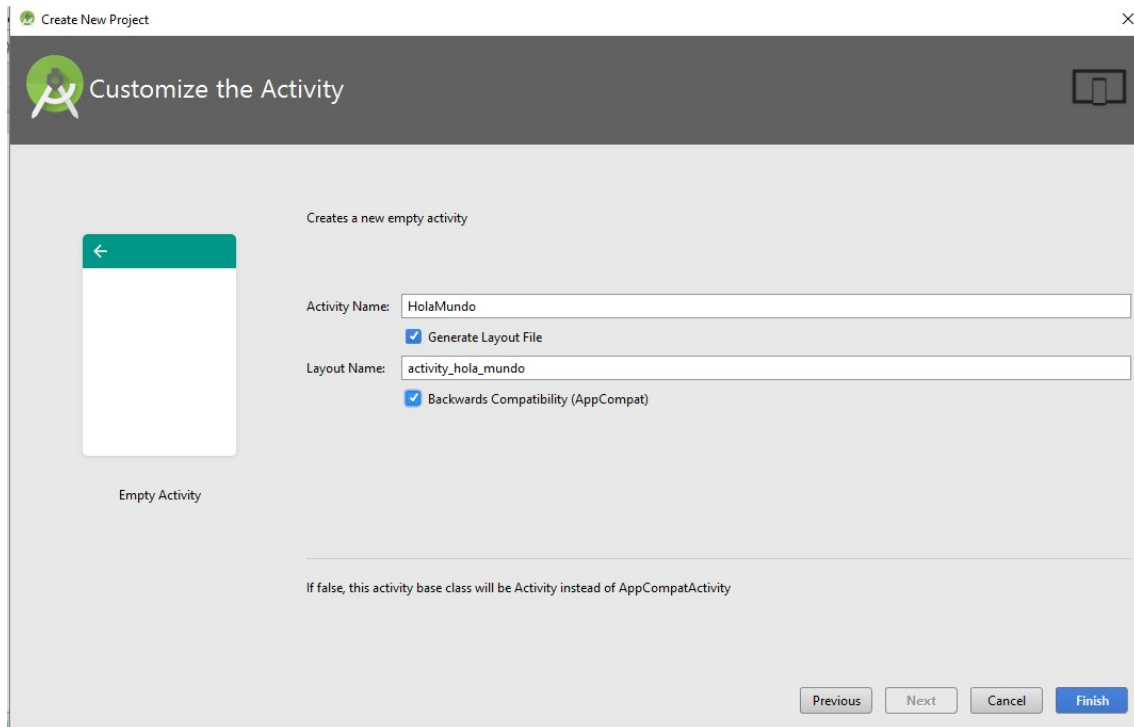
Minimum SDK API 21: Android 5.0 (Lollipop)

☐ Android Auto

- Pulsamos Next.
- Ventana "Add on Activity to Mobile" "tema" (aspecto visual) de la aplicación. seleccionamos Empty Activity.
- Pulsa Next.



- **Activity Name:** Nombre que vamos a poner a la clase principal, "HolaMundo".
- Casilla Generate Layout File.
- **Layout Name:** Si queremos crear el fichero layout, aquí nos aparece el nombre de este fichero.
- Pulsa Finish



Aunque no vamos a entrar en mucho detalle, aún, con respecto a todo lo que el asistente ha creado por nosotros, hay algunas cosas que podemos apreciar en los ficheros que ha añadido a nuestro recién creado proyecto:

- **java/paquete:** es el directorio con el código fuente de nuestras clases. Por el momento únicamente hay una, `HolaMundo.java`, que resulta ser la actividad que le pedimos que creara por nosotros.
- **manifests:** directorio con el fichero `AndroidManifest` es el fichero controlador de actividades.
- **res/drawable:** carpeta para poner las imágenes que necesitemos.
- **res/layout:** guarda los ficheros layout "distintas ventanas de nuestra aplicación".
- **res/mipmap:** guarda los iconos de la aplicación.
- **res/values:** en este directorio aparecen los ficheros de colores, dimensiones, variables de tipo texto, estilos.
- **Gradle Scripts:** aquí hay varios ficheros de configuración, en estos momentos solamente nos interesa uno "build.gradle (Module app)"

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 25
    buildToolsVersion "25.0.2"
    defaultConfig {
        applicationId "da2d1.puzzle_yineth"
        minSdkVersion 16
        targetSdkVersion 25
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner
        "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
```

```

        minifyEnabled false
        proguardFiles getDefaultProguardFile('proguard-
android.txt'), 'proguard-rules.pro'
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    androidTestCompile('com.android.support.test.espresso:espresso-
core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-
annotations'
    })
    compile 'com.android.support:appcompat-v7:25.0.1'
    testCompile 'junit:junit:4.12'
}

```

Práctica 2.2: Interactuando con el usuario: botón "Púlsame"

Vamos a crear nuestra segunda aplicación, que contendrá un botón que podremos pulsar.

```

public class Pulsame extends Activity {

    /**
     * Método llamado cuando se crea la actividad.
     *
     * @param savedInstanceState Ignoramos el parámetro.
     */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        // Llamamos al método que estamos sobrescribiendo
        // de la clase padre.
        super.onCreate(savedInstanceState);

        // Configuramos la ventana, añadiendo un botón
        // que llamará a nuestro método protegido.
        _boton = new Button(this);
        _boton.setText("¡Púlsame!");
        _boton.setOnClickListener(
            new View.OnClickListener() {
                public void onClick(View v) {
                    botonPulsado();
                }
            }
        );
        // Ponemos el botón como componente de la
        // actividad.
        setContentView(_boton);
    } // onCreate

    //-----

    /**
     * Método llamado cuando se pulsa sobre el botón

```

```

    * de la ventana. Es llamado a través de la clase
    * anónima del evento.
    */
private void botonPulsado() {

    // Incrementamos el contador...
    ++_numVeces;

    // ... y actualizamos la etiqueta del botón.
    _boton.setText("Pulsado " + _numVeces + " veces");

} // botonPulsado

//-----
//                      Atributos protegidos/privados
//-----

/**
 * Botón de la ventana.
 */
private Button _boton;

/**
 * Número de veces que se ha pulsado el botón.
 */
private int _numVeces;

} // Pulsame

```

- Lanza la aplicación tanto en el emulador API16 como en otro con API menor de 14 y pruébala. Fíjate que también queda "instalada".

Práctica 2.3: Nivel de API mínimo y de construcción

En las prácticas anteriores, el asistente nos preguntó por el mínimo SDK requerido, el target SDK¹ y la versión con la que compilar.

Cuando compilamos una aplicación necesitamos tener acceso a las clases auxiliares que vamos a utilizar, como por ejemplo la clase padre de las actividades, `android.app.Activity`, de la que hereda la única clase de nuestro programa. Si Android Studio no encuentra esa clase, no podrá compilar el programa.

Esas clases las proporciona la plataforma Android contra la que estamos compilando nuestra aplicación (Compile with). A modo de ejemplo, si en cualquiera de los proyectos anteriores despliegas `android.jar` (en la "carpeta" `Android <version>`), verás que, entre otras muchas, está la mencionada `android.app.Activity`. Las versiones más recientes de la plataforma contienen las clases de las versiones anteriores, junto con otras nuevas que proporcionan funcionalidad adicional.

Es importante ser consciente de que cuando enviamos una aplicación a un dispositivo (el `.apk` que nos ha generado Android Studio), esas clases no se incluyen. Dicho de otro modo, se espera que el dispositivo disponga del fichero `android.jar`. Pero, ¿qué versión? Esa pregunta la responde Minimum Required SDK. En el caso de

¹ En realidad, el término SDK aquí es incorrecto. Debería decir API, dado que nos está preguntando por la versión de la plataforma Android, no de las herramientas usadas para el desarrollo, que llevan una numeración diferente.

nuestro ejemplo, esperamos que al menos el dispositivo disponga de (API 14), aunque podría ser superior. Es muy importante especificar el mínimo posible, dado que si publicáramos nuestra aplicación en el "market", los dispositivos con versiones de Android inferiores a la mínima solicitada no verán nuestro programa (y no podrán instalarla). Poner un valor caprichosamente alto, por tanto, nos restará visibilidad.

El punto negativo de tener los dos valores es que debemos ser cuidadosos al desarrollar, para no utilizar características de la versión Compile with que no estén en Minimum Required SDK. Si lo hacemos, fallará en ejecución en el dispositivo.

Afortunadamente, las SDK contienen una herramienta llamada Lint que analiza el proyecto y avisa de construcciones erróneas o potencialmente problemáticas. Android Studio lo ejecuta automáticamente, por lo que si hacemos uso de alguna funcionalidad que no se soporta en el Minimum Required SDK, recibiremos un error de lint.

Práctica 2.4: Ejecución en un dispositivo físico

Si queremos ejecutar nuestra aplicación en un dispositivo físico, tendremos que conectarlo al ordenador por USB. En Windows necesitaremos instalar los drivers que se proporcionan con las SDK, en uno de los paquetes de la sección Extra². En GNU/Linux normalmente no es necesario instalar nada.

Normalmente los móviles no permiten la ejecución de aplicaciones y el control externo. La conexión por USB es utilizada por la mayoría de los usuarios para sincronización de datos y acceso al almacenamiento, no para ejecutar aplicaciones adicionales. Por tanto, antes de conectar el dispositivo por USB es necesario configurarlo para que admita lo que se denomina depuración por USB. Para eso, ve a los Ajustes, Aplicaciones, Desarrollo, y marca Depuración de USB.

Cuando lo hagas, conecta el móvil por USB. Si todo va bien, las SDK de Android deberían tener acceso a él. Para comprobarlo, puedes usar una aplicación en consola:

```
$ cd $ANDROID_SDK_PATH
$ cd platform-tools
$ ./adb devices
List of devices attached
emulator-5554 device
emulator-5556 device
D32C76402AF837AE device
```

En este caso, vemos los dos AVDs lanzados, y un dispositivo físico conectado por USB. Si no ves tu móvil, comprueba que tienes los controladores necesarios.

Para lanzar una aplicación sobre el móvil, basta con que en Android Studio inicies una ejecución normal. Android Studio te mostrará la lista de los dispositivos disponibles que pueden ejecutar tu programa, y es suficiente con que elijas el dispositivo físico. Verás tu aplicación en el móvil un instante después.

Ten en cuenta que la aplicación quedará instalada, y podrás utilizarla posteriormente incluso aunque desconectes el móvil y deshabilites la depuración por USB.

- Recupera el proyecto de la práctica 2.2 y ejecútala en tu móvil.

² Dependiendo del móvil, podrían necesitarse controladores específicos del fabricante.

Práctica 2.5: LogCat: el registro en Android

En las aplicaciones para móviles funcionan los habituales `System.out` o `System.err`, dado que no hay consola no deberíamos utilizarlas. Puedes probarlo si añades en el programa de la práctica 2.2, al final del método `botonPulsado()` el código:

```
System.out.println("Pulsado " + _numVeces + " veces");
```

Si lo ejecutas, verás que en la pestaña **Logcat** (línea inferior de Android Studio) aparece la siguiente salida.

```
2022-09-26 10:05:06.389 4886-4886/dm2e.luis.practica23 I/System.out: Pulsado 1 veces
2022-09-26 10:05:06.599 4886-4886/dm2e.luis.practica23 I/System.out: Pulsado 2 veces
2022-09-26 10:05:06.809 4886-4886/dm2e.luis.practica23 I/System.out: Pulsado 3 veces
2022-09-26 10:05:06.979 4886-4886/dm2e.luis.practica23 I/System.out: Pulsado 4 veces
```

Como ayuda a la depuración, en Android tenemos el sistema de log llamado LogCat. Las aplicaciones envían mensajes al sistema de log, y, durante la ejecución, Android Studio obtiene todos los mensajes y nos los muestra en la ventana LogCat.

Los mensajes de registro tienen la siguiente información:

- **Tag:** es una cadena elegida por la aplicación que ha generado el mensaje, y que lo identifica. Normalmente se utiliza el nombre del programa. El sistema Android envía sus propios mensajes de registro, y utiliza tags que identifican los componentes internos que los han generado (`ActivityManager`, `dalvikvm`, `AndroidRuntime`, etcétera).
- **Nivel:** es el nivel de prioridad (gravedad) del mensaje:
 - ✓ *Verbose:* mensajes de poca importancia, con detalles extremadamente finos de la ejecución. Nunca deberían generarse mensajes con esta prioridad en una aplicación en producción.
 - ✓ *Debug:* mensajes de depuración.
 - ✓ *Information:* mensajes informativos sobre la ejecución.
 - ✓ *Warning:* avisos de sucesos importantes que podrían terminar desembocando en problemas.
 - ✓ *Error:* notificaciones de errores que afectan a la aplicación.

Para enviar mensajes al log, utilizamos la clase `android.util.Log`, y sus métodos `v()`, `d()`, `i()`, `w()` y `e()`, para cada uno de las cinco prioridades de errores. Todos son estáticos, y reciben dos parámetros. El primero es el tag, que identifica a la aplicación; el segundo es el mensaje que queremos registrar.

- Recupera la práctica 2.2, y añade un atributo estático:

```
/**
 * Constante con el "tag" usado para registro de sucesos
 * en LogCat relativos a esta actividad.
 */
private static final String TAG = "Pulsame";
```

- Al final del método `botonPulsado()` añade el código:

```
android.util.Log.d(TAG, "botonPulsado " + _numVeces);
```

- Lanza la aplicación en alguno de los AVD.
- Ve a la ventana LogCat de Android Studio. Si no está visible, ábrela con Window, Show View, LogCat. Observa la cantidad de mensajes que aparecen, relacionados con el propio sistema Android.
- Pulsa el botón varias veces.
- Comprueba la aparición del mensaje en el LogCat.
- Ejecuta la aplicación en el dispositivo físico y comprueba la salida de LogCat.

Práctica 2.6: Android Debug Bridge

En el capítulo 1 vimos que cada AVD tenía un identificador equivalente a un número de puerto TCP en el que se quedaba escuchando para recibir solicitudes a través de un protocolo de línea de comandos. Esos números de puerto siempre son pares.

Además, cada AVD escucha en el puerto impar inmediatamente posterior (empezando por 5555) y proporciona un servicio de depuración que sigue un protocolo particular comprendido por el demonio adb.

Ese protocolo no se basa en texto, pero podemos aprovecharnos de él usando la aplicación adb.

El esquema de uso es el siguiente:

- **adb devices:** muestra los dispositivos detectados.
- **adb [-d | -e | -s <numeroSerie>] comando:** ejecuta el comando sobre el dispositivo solicitado.
 - ✓ **-d:** al único dispositivo conectado por USB. No podrá ser usado si hay más de uno.
 - ✓ **-e:** al único emulador detectado. No podrá ser usado si hay más de uno.
 - ✓ **-s <numeroSerie>** al dispositivo indicado.
- Conéctate a cualquiera de los AVDs que tengas lanzados, y muestra el logcat. Por ejemplo, asumiendo que sólo hay un AVD:

```
$ ./adb -e logcat
```

- La salida que muestra tiene menos información que la vista en Android Studio. Muestra únicamente la gravedad, el tag, y el PID. Para mostrar toda la información, prueba:

```
$ ./adb -e logcat -v threadtime
```

Ahora verás toda la información que nos daba Android Studio. Las alternativas son brief, process, tag, raw, time, threadtime y long. Pruébalas e identifica sus diferencias.

- Prueba a poner filtros:

```
$ ./adb -e logcat -v ActivityManager:* dalvikvm:* *:S
```

- Abre un shell con algún AVD. Por ejemplo, si sólo tienes uno lanzado:

```
$ ./adb -e shell
```

```
# <<--- shell remoto en el móvil
```

- Observa el #, que es el prompt del root.

- Utiliza los comandos del shell habituales en GNU/Linux para recorrer la estructura de directorios. Usa ps para ver los procesos lanzados o top para ver el consumo (aunque no es tan ágil como el tradicional).
- Termina saliendo con exit.

Si tienes un dispositivo físico, podrás hacer lo mismo accediendo a él (si habilitaste la depuración de programas). Haz las pruebas de logcat y también las del shell. Observa que en ese caso no serás root (el prompt que aparece es \$).

Práctica 2.7: Ciclo de vida de las aplicaciones

- Lanza la aplicación Pulsame en cualquiera de los AVDs. Pulsa varias veces el botón, para que aumente el número.
- Pulsa "escape" o "Atrás".
- Vuelve a lanzar el programa, buscándolo entre las aplicaciones instaladas.
- Observa que se reinicia la cuenta. El programa se cerró, y se ha lanzado de nuevo desde el principio.
- Pulsa varias veces sobre el botón otra vez.
- Simula una llamada entrante en el dispositivo:
 - ✓ Conéctate por telnet al puerto del AVD indicado en el título de la ventana:

```
$ telnet localhost 55xx
```

- ✓ Simula una llamada entrante de un número arbitrario:

```
gsm call 555666777
```

- Descuelga la llamada en el AVD, y luego finalízala.
- Terminarás volviendo a nuestra aplicación que esta vez no se habrá reiniciado.
- Pulsa ahora la tecla Inicio, o el botón "Home", para volver al "escritorio" de Android.
- Lanza de nuevo el programa, buscándolo entre las aplicaciones.
- Aunque lo hayamos "lanzado" otra vez, no se ha reiniciado, y la cuenta se mantiene.

Fíjate que el programa que hemos hecho no tiene método main(), igual que ocurre en los applets. De hecho, nuestra "aplicación" hereda de la clase android.app.Activity. Una actividad es un componente de una aplicación, que muestra una ventana con la que interactuar (buscar en la agenda, hacer una foto, escribir un mensaje, mirar los mensajes entrantes). Cada actividad tiene una ventana que, normalmente, ocupará toda la pantalla.

El ciclo de vida de las actividades lo controla el propio Android. Es el sistema quién decide cuando eliminar una actividad. Hemos visto que:

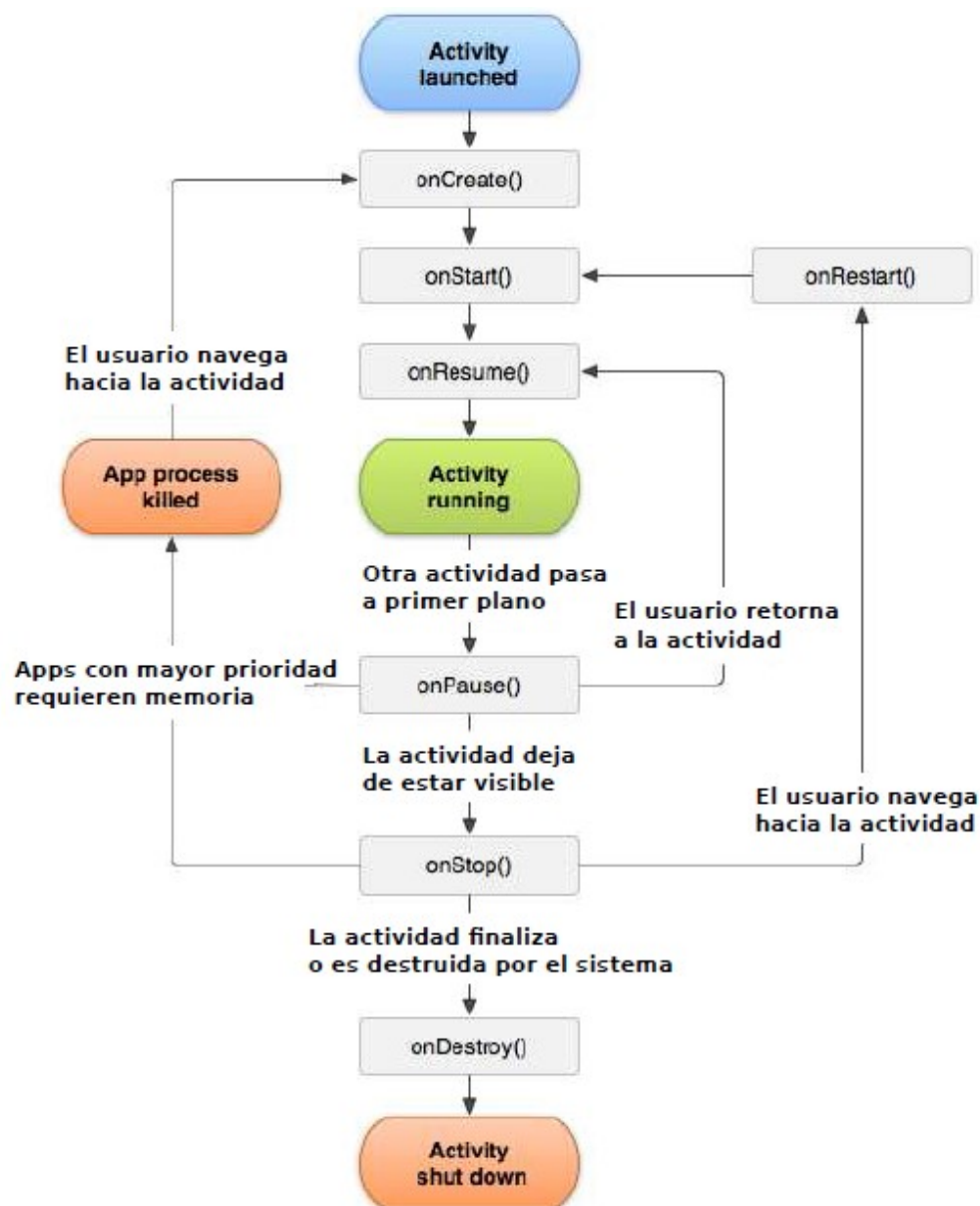
- Al pulsar "Atrás", el sistema elimina la actividad, dado que al volverla a ejecutar se reinició la cuenta.
- Si se superpone una aplicación nueva sobre la nuestra (como ocurre cuando recibimos una llamada), la actividad no es destruida, y volveremos a ella al terminar. Android mantiene una pila de actividades.

- Si pulsamos el botón "Home", la actividad se mantiene lanzada, pero no es visible, ni podemos volver a ella a no ser que la "lancemos" de nuevo. Android detecta que ya tiene una instancia abierta, y vuelve a ella.

Igual que en los applets, podemos sobrescribir métodos de la superclase `android.app.Activity` para ser notificados de cambios en el estado. El ciclo de vida se resume en la siguiente figura:

Los métodos llamados por el sistema aparecen en rectángulos. Deberías reconocer `onCreate()`, dado que es donde en la práctica 2.2 creábamos y configurábamos el botón. Siempre que se sobrescribe un método es obligatorio llamar al método de la clase padre; de otro modo se generará una excepción en ejecución.

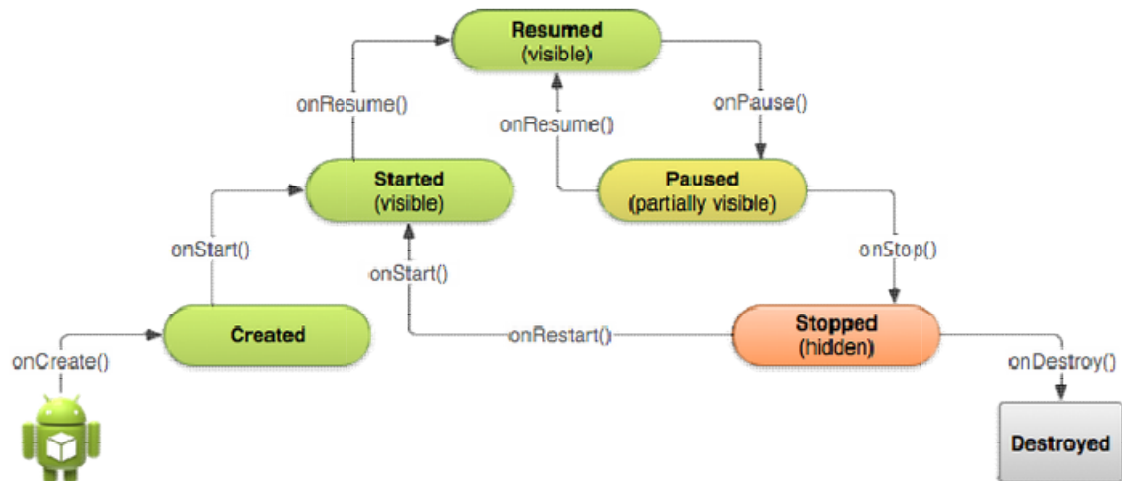
Una forma diferente de representar el ciclo de vida es como una máquina de estados, donde los nodos son los estados y las transiciones representan los eventos, es decir los métodos a los que invoca el sistema.



Ten en cuenta, que los estados Created y Started son transitorios; el sistema pasará por ellos rápidamente invocando a los siguientes eventos.

Vamos a hacer una nueva aplicación que nos muestre un mensaje cuando se invoque a los métodos que controlan el ciclo de vida.

- Crea un nuevo proyecto Android:
 - Application Name: Ciclo de vida
 - Project name: CicloDeVida



```
public class CicloDeVida extends Activity {

    /**
     * Método invocado por Android cuando lanza la actividad.
     * Inicializamos el interfaz gráfico.
     *
     * @param savedInstanceState Lo ignoramos.
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_ciclo_de_vida);
        android.util.Log.i(TAG, "onCreate");
    }

    //-----

    /**
     * Método llamado cuando la actividad pasa de estar
     oculta a
     * estar, al menos, parcialmente visible.
     */
    protected void onStart() {
        super.onStart();
        android.util.Log.i(TAG, "onStart");
    }

    //-----

    /**
     * Método llamado cuando se recupera una actividad
     después
```

```

        * de haber estado detenida.
        */
protected void onRestart() {
    super.onRestart();
    android.util.Log.i(TAG, "onRestart");
}

//-----

/**
 * Método llamado cuando la actividad pasa a primer
plano.
 */
protected void onResume() {
    super.onResume();
    android.util.Log.i(TAG, "onResume");
}

//-----

/**
 * Método llamado cuando la actividad deja de estar
 * en primer plano (puede que siga siendo parcialmente
 * visible).
 */
protected void onPause() {
    super.onPause();
    android.util.Log.i(TAG, "onPause");
}

//-----

/**
 * Método llamado cuando la actividad se detiene
 * (deja de estar visible).
 */
protected void onStop() {
    super.onStop();
    android.util.Log.i(TAG, "onStop");
}

//-----

/**
 * Método llamado antes de que la aplicación se destruya.
 */
protected void onDestroy() {
    super.onDestroy();
    android.util.Log.i(TAG, "onDestroy");
}

//-----
//          Atributos protegidos/privados
//-----

/**

```

```

        * Constante con la etiqueta que nos identificará en el
        * logcat.
        */
private static final String TAG = "CicloDeVida";

} // class CicloDeVida

```

- Ejecútalo en cualquier AVD, y repite las pruebas anteriores, mirando la ventana LogCat en Android Studio.
- Ejecútalo en un dispositivo físico y pruébalo también. Espera a que el dispositivo se suspenda, con la aplicación en primer plano, y mira el registro en Android Studio. Reactiva el dispositivo y comprueba el resultado en el log.

Práctica 2.8: Primera aproximación a los recursos: internacionalización

En la práctica 2.2 creamos una aplicación en la que establecíamos el texto del botón directamente por código. Eso no es buena idea, porque impedimos la internacionalización de nuestra aplicación.

Las cadenas de texto son un tipo especial de recursos, que el sistema puede gestionar automáticamente por nosotros. En código le pedimos que haga uso de un recurso a partir de su identificador, y Android se encargará de buscar el valor más adecuado para la configuración actual del sistema.

- Haz una copia del proyecto Pulsame y ponle el nombre PulsameLocalizado. Vamos a cambiarlo para hacer uso de recursos de tipo cadena, y haciendo una copia mantenemos la versión inicial. No obstante, como el nombre del paquete no lo cambiaremos, no podremos tener en el mismo dispositivo las dos aplicaciones a la vez.
- Abre el fichero de localización de cadenas en `res/values/strings.xml`. Verás que hay varias cadenas definidas, que nos creó el asistente al crear el proyecto. Podemos utilizar el interfaz gráfico para modificarlas, o podemos abrir directamente el fichero en su forma XML, en la pestaña inferior. Añade una nueva cadena. Para eso, pulsa Add, String. Como identificador pon `pulsameAdmiracion` y como texto `!Púlsame!`. En la vista XML, observa que tu nueva cadena se ha incluido.
- Abre el fichero `R.java` en la carpeta `gen/` del proyecto. Es un fichero generado automáticamente a partir de los recursos. Observa la aparición del campo `R.string.pulsameAdmiracion`, al que se le ha asignado un valor numérico arbitrario.
- En el código fuente, en el método `onCreate`, sustituye la línea:

```
_boton.setText("!Púlsame!");
```

por:

```
_boton.setText(R.string.pulsameAdmiracion);
```

- Ejecuta el programa. No deberías notar ninguna diferencia visible. Pero hemos dado el primer paso hacia la localización.
- Vamos a crear ahora un nuevo fichero de cadenas para que se utilice cuando el móvil esté configurado con idioma inglés. Selecciona la carpeta `res/`, y en el menú contextual pulsa New, Other (o pulsa Ctrl-N).
- Selecciona Android - Android XML Values File.

- Especifica strings.xml como nombre. Nos avisa de que el fichero ya existe en el directorio res/values donde debería ir para que Android lo considere un fichero de recurso de cadenas. Sin embargo es un mero aviso, dado que todavía podemos añadirle cualificadores. Pulsa Next.
- Añade como cualificador Language, y establece en (inglés). El mensaje de aviso de fichero ya existente desaparece. Nos indica que el nuevo fichero se guardará en res/values-en.
- En el nuevo fichero, que de momento no tiene cadenas, crea una nueva.
- Como identificador escribe pulsameAdmiracion (el mismo que pusimos antes), y como texto Push me!.
- Relanza la aplicación. Verás que el texto del botón aparece ahora en inglés. Sin embargo, el nombre de la aplicación sigue saliendo en español; no hemos localizado todas las cadenas, y se hace uso de los valores por defecto (en el directorio sin cualificadores).
- Añade al nuevo fichero de cadenas (en inglés), las que teníamos en español. Es más rápido utilizar la vista en XML, copiar y pegar el contenido original y modificar lo que corresponda. Verás que hay algunas cadenas que añadió el asistente que ni siquiera estamos usando (y que de hecho aparecen en inglés). Lanza la aplicación otra vez y comprueba que todo está en inglés, incluso el nombre de la aplicación en el lanzador.
- Cambia la configuración del idioma del AVD a español. Para eso, ve a Settings, Language & keyboard (o Locale & text en algunas versiones), Select language, y elige Español (España).
- El interfaz del AVD te aparece ahora en español. Busca la aplicación y ejecútala. Comprueba que aparece en español.
- Renombra el directorio values a values-es, para indicar que debe utilizarse cuando el idioma sea español. Relanza la aplicación desde Android Studio para que se envíe al móvil la nueva versión. No deberías notar ningún cambio.
- Modifica la configuración del AVD y ponlo en francés.
- La aplicación falla al no encontrar los recursos. Es muy importante tener siempre valores predefinidos de los recursos para que se usen en las configuraciones de móviles que no tengan versiones específicas de ellos.
- Cambia de nuevo el nombre del directorio values-es a values. Vuelve a poner el idioma en español.

Tradicionalmente, un problema de la localización/internacionalización ha sido los plurales contruídos en código, pues cada idioma los crea de una forma. Afortunadamente, Android también nos da soporte para ellos aunque el ADT no nos proporcione ninguna ayuda.

- Entra en la vista XML de strings.xml en español y añade:

```
<plurals name="numPulsaciones">
    <item quantity="one">Pulsado %d vez</item>
    <item quantity="other">Pulsado %d veces</item>
</plurals>
```

Esto es un nuevo recurso de nombre numPulsaciones, que dice que cuando la cantidad sea 1 debe usarse la primera cadena, y cuando sea un número diferente, la segunda.

- En el fichero strings.xml para el inglés pon:

```
<plurals name="numPulsaciones">
```

```

        <item quantity="one">Pushed %d time</item>
        item quantity="other">Pushed %d times</item>
    </plurals>

```

Observa que el identificador es el mismo, pero han cambiado los textos.

- Necesitamos internacionalizar nuestra aplicación, para hacer uso de este nuevo recurso, que ahora es más complicado. Sustituye el código del método `botonPulsado()` por:

```

private void botonPulsado() {
    ++_numVeces;
    android.content.res.Resources res = getResources();
    String numPulsados;
    numPulsados = res.getQuantityString(
        R.plurals.numPulsaciones,
        _numVeces,
        _numVeces);
    _boton.setText(numPulsados);
} // botonPulsado

```

Lo que hacemos es conseguir el gestor de recursos, y pedirle la cadena de plurales cuyo identificador es `R.plurals.numPulsaciones` especifica para cuando el número es `_numVeces` (segundo parámetro), y que nos sustituya el `%d` que incluye por el valor de `_numVeces` (tercer parámetro).

- Ejecuta la aplicación tanto con la configuración en español como en inglés, y observa que se utiliza la forma correcta.

Práctica 2.9: Interfaces de usuario como XML

En la práctica 2.8 hemos hecho uso de recursos para las cadenas, que ya no están cableadas en el código. Sin embargo, estamos construyendo el interfaz de la ventana manualmente. El método `onCreate()` es:

```

public void onCreate(Bundle savedInstanceState) {
    // Llamamos al método que estamos sobreescribiendo
    // de la clase padre.
    super.onCreate(savedInstanceState);

    // Configuramos la ventana, añadiendo un botón
    // que llamará a nuestro método protegido.
    _boton = new Button(this);
    _boton.setText(R.string.pulsameAdmiracion);
    _boton.setOnClickListener(
        new View.OnClickListener() {
            public void onClick(View v) {
                botonPulsado();
            }
        }
    );
    // Ponemos el botón como componente de la
    // actividad.
    setContentView(_boton);
} // onCreate

```

- Llamamos al método de la superclase que estamos sobrescribiendo.
- Creamos un nuevo objeto Button, que representa un control (widget) de usuario.
- Establecemos su texto utilizando un recurso, que puede ser localizado.
- Añadimos un listener al botón, para que sea invocado cuando el usuario lo pulse. Usamos una función anónima que invocará al método botonPulsado() de la clase.
- Por último, le decimos a la actividad (this) que como contenido establezca al botón que hemos creado, que ocupará toda la ventana.

Dado que el botón lo estamos guardando en el campo `_boton`, en el método `botonPulsado()` podemos acceder a él para modificar el texto que muestra, tal y como ya hemos visto.

Configurar los controles de la ventana manualmente por código no es buena idea. Salvo para interfaces simples, resulta bastante laborioso, y cada prueba requiere recompilación y ejecución, siendo un proceso lento. Además si quisiéramos personalizar el interfaz para diferentes entornos, igual que hemos hecho con las cadenas, tendríamos que llenar nuestro programa de código condicional. Por ejemplo, podríamos querer que el interfaz cambiara si el dispositivo está en vertical (portrait) o en horizontal (landscape), o que en una ventana pidiendo datos personales pidiera el segundo apellido en España, pero no en Estados Unidos, donde debería pedir a cambio el middle name.

Afortunadamente, Android dispone de un tipo de recursos llamados layout que permiten especificar la disposición de los controles de un interfaz de usuario (una actividad) en un XML. Como con las cadenas, ese XML tendrá un identificador, y podremos pedirle al sistema que nos construya el interfaz a partir de él. Podremos tener diferentes versiones del mismo layout, cualificadas con las opciones del sistema de una forma similar a la que ya vimos para las cadenas.

Vamos a modificar poco a poco nuestra aplicación para hacer uso de un layout como recurso.

- Haz una copia del proyecto PulsameLocalizado y dale un nuevo nombre, por ejemplo PulsameLayout.
- Elimina (o comenta) todo el código del método `onCreate()`.
- Busca el fichero `res/layout/activity_pulsame.xml`. Es un fichero que nos creó el asistente del proyecto, y que no estábamos usando, al haber sustituido el `onCreate()`. Contiene el recurso predefinido de la actividad que nos escribió para que lo usáramos como punto de partida. Ábrelo. Si se te abre el Graphical Layout, selecciona la vista en XML en la pestaña inferior.
- Elimina todo el texto que aparece, y copia el siguiente:

```
<Button
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/boton"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:text="@string/pulsameAdmiracion"/>
```

Estamos indicando que nuestro layout (disposición) contendrá únicamente un objeto de la clase Button. El primer atributo es estándar para especificar el espacio de nombres XML. El resto de campos son más interesantes:

- ✓ **android:id:** indica el identificador del botón. Podremos utilizarlo para acceder al botón desde código. El identificador de los controles siempre deben tener la estructura `@id/<identificador>` o `@+id/<identificador>` (la diferencia está en el +). Si no se pone el +, tendremos que definir nosotros el identificador de alguna forma en el fichero R (lo normal será porque estemos reutilizando un identificador). Si se indica el + (lo más habitual), le pedimos a apt (la herramienta que procesa los recursos) que nos añada el identificador automáticamente a la clase R.
- ✓ **android:layout_width** y **android:layout_height:** indica cuánto espacio del contenedor en el que se muestra el control (en este caso, el botón en la ventana) deberá ocupar a lo ancho y a lo alto. Los posibles valores son:
 - **match_parent** (o **fill_parent**): el control debe ocupar todo el espacio disponible para él en el contenedor padre.
 - **wrap_content:** el control ocupará el espacio mínimo necesario para mostrarse.

En este caso, hemos elegido ocupar el máximo espacio, tanto a lo ancho como a lo alto.

- ✓ **android:text:** texto del botón. Si lo comenzamos con `@`, Android asumirá que estamos accediendo a un recurso, y buscará la cadena asociada a él. Si no, pondrá el texto que escribamos. Lo recomendable es usar siempre recursos.
- Android Studio automáticamente nos regenera la clase R con los identificadores de los recursos. Ve a ella, y comprueba que hay dos nuevos:
 - ✓ **R.layout.activity_pulsame:** el nombre proporcionado para este recurso es el nombre del fichero (sin la extensión). En el caso de las cadenas, el nombre se especificaba en el XML, y el nombre del fichero era indiferente. Dado que los layouts son recursos que ocupan un fichero completo, el identificador que se utiliza es el del nombre del fichero.
 - ✓ **R.id.boton:** es el identificador del botón del layout. Lo ha creado automáticamente al especificar el + en él.
- Ya tenemos definido el recurso. Ahora tenemos que usarlo. En el método `onCreate()`, comenta (o elimina) todo el código salvo la llamada al método de la superclase, y añade:

```
setContentView(R.layout.activity_pulsame);
Recuerda que originalmente teníamos:
setContentView(_boton);
```

donde `_boton` era el objeto que habíamos creado manualmente. Ahora le estamos diciendo a Android "como contenido, pon los controles definidos en el recurso asociado a este identificador". Android buscará el recurso, creará y configurará los objetos a partir de los datos del XML, y lo asociará al contenido de la actividad.

- Ejecuta la aplicación y comprueba que todo va bien. Deberías ver el texto en el botón (con el idioma localizado), aunque no hará nada al ser pulsado.
- Modifica el layout y prueba a establecer `wrap_content` en lugar de `fill_parent` en `layout_*` para ver cómo funciona.
- Para poder reaccionar al evento, necesitamos configurar el botón. Para eso, necesitamos acceder a él, e invocar a su `setOnClickListener()`, como hacíamos con el botón creado manualmente. Para buscar un control por el identificador de su recurso, se utiliza `findViewById(id)`, que devuelve un objeto de la clase

android.view.View, padre de todos los controles. Añade el siguiente código al método onCreate():

```
// Obtenemos el objeto Button a partir de su id.
_boton = (Button) findViewById(R.id.boton);
// Nos hacemos observadores de sus pulsaciones.
_boton.setOnClickListener(
    new View.OnClickListener() {
        public void onClick(View v) {
            botonPulsado();
        }
    }
);
```

Ahora el campo _boton se está obteniendo del recurso (una vez que ya se ha configurado la vista), en lugar de crearlo manualmente como hacíamos antes con new. Además, ya no establecemos el contenido (lo hace el propio layout). Únicamente añadimos el listener.

- Prueba la aplicación. Verás que vuelve a funcionar la pulsación del botón.
- Si te fijas en el método onClick() que estamos escribiendo en la clase anónima que captura el objeto, verás que tiene un parámetro View v. El objeto que recibimos ahí es el widget que ha ocasionado la invocación al método, es decir el propio botón que ha sido pulsado. Aunque el prototipo del método dice que recibe un objeto de la clase View, en realidad eso es el tipo estático, pero el tipo dinámico (gracias al polimorfismo) es en realidad Button. Vamos a modificar el código para que desde la clase anónima enviemos ese objeto al método botonPulsado(). Se lo enviaremos aún con tipo (estático) View, y haremos la conversión de tipos (_hacia abajo_) en el propio botonPulsado(). Es decir, el código debería pasar ahora a ser:

```
// ...
public void onCreate(Bundle savedInstanceState) {
    ....
    // Nos hacemos observadores de sus pulsaciones.
    _boton.setOnClickListener(
        new View.OnClickListener() {
            public void onClick(View v) {
                botonPulsado(v);
            }
        }
    );
} // onCreate
...
private void botonPulsado(View v) {
    ++_numVeces;
    android.content.res.Resources res = getResources();
    String numPulsados;
    numPulsados = res.getQuantityString(R.plurals.numPulsaciones,
        _numVeces, _numVeces);
    Button b = (Button) v;
    b.setText(numPulsados);
} // botonPulsado
```

La diferencia principal es que ahora en `botonPulsado()` no necesitamos el campo `_boton`, dado que lo recibimos directamente como parámetro (aunque nos veamos forzados a realizar una conversión de tipos).

- Prueba la aplicación y comprueba que sigue funcionando.
- Dado que ya no necesitamos `_boton` en `botonPulsado()`, podemos ahorrárnoslo completamente. Elimina el atributo, y en `onCreate()` crea una variable local que sea un `Button` para guardar el valor devuelto por `findViewById()`.

Necesitar acceder al widget para establecer el listener que nos llame al método de procesamiento de la pulsación es bastante tedioso. Es posible especificarlo directamente en el recurso, y que Android haga la asociación automáticamente.

- En la definición del layout, añade un nuevo atributo al botón:

```
android:onClick="botonPulsado"
```

- En el código fuente del método `onCreate()`, elimina la parte de establecer el listener. El código completo del método deberá ser mucho más corto:

```
public void onCreate(Bundle savedInstanceState) {  
    // Llamamos al método que estamos sobrescribiendo  
    // de la clase padre.  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_pulsame);  
}
```

- Convierte en `public` el método `botonPulsado()`.
- Prueba la aplicación.

Para que esto funcione, el método de la actividad establecido en el atributo `android:onClick` del XML debe cumplir:

- Ser público
- No devolver ningún valor (`void`)
- Recibir un único parámetro de tipo `View`, que contendrá el elemento pulsado.

Por desgracia, no todos los eventos de los controles se pueden asignar directamente a través del XML.

Práctica 2.10: Propiedades básicas de los controles

En la práctica 2.9 definimos nuestro primer interfaz de usuario con un único botón, que, además, respondía a su pulsación.

Antes de adentrarnos en disposiciones más complicadas, con varios controles, vamos a familiarizarnos con algunas de las propiedades visuales más importantes de los controles. La propiedad `android:id` es importante para la definición de un identificador, aunque no tiene repercusión visual. Por su parte, la propiedad `android:text` de los botones especifica el texto que muestra, y tampoco nos detendremos más en él.

Para las pruebas que vamos a realizar, abre el fichero del recurso de tipo layout `activity_pulsame.xml` de dicha práctica usando, esta vez sí, la vista Graphical Layout. Android Studio nos muestra un entorno gráfico para diseñar la ventana, en lugar de tener que utilizar directamente el XML. Nos permite así hacer cambios y ver el

resultado automáticamente, sin necesidad de compilar la aplicación y probarla en un AVD o terminal físico.

Es interesante que según vayas realizando las pruebas, modifiques la orientación de la pantalla, para probar cómo quedaría en una posición apaisada del dispositivo:

- Fíjate que los atributos que conocíamos con los nombres `layout_width` y `layout_height` en el XML, están, en el editor de propiedades, dentro de una sección Layout Parameters y se llaman, sencillamente, `width` y `height`. Esto es debido a que son atributos "heredados" del contenedor en el que está el botón, no del botón en sí mismo. En el XML tendremos que especificar como nombre `layout_*`, pero en la vista gráfica tenemos que buscarlos dentro de esa sección de las propiedades.
- Como hemos visto, `layout_width` sirve para indicar cuánto espacio horizontal del contenedor donde está el control ocupa éste. Con el valor `match_parent` (o `fill_parent`) se ocupa todo el espacio posible. Cámbialo a `wrap_content` y observa el resultado.
- Haz lo mismo con `layout_height`.
- También en esa misma sección podemos ver la propiedad `gravity` (que será `layout_gravity` en el XML). Experimenta con ella. ¿Averiguas para qué sirve? Alfinal, déjala vacía.
- Vuelve a colocar en `layout_*` en `fill_parent`. Manipula la propiedad `margins` del `layout parameters` (en el XML será `layout_margin` para el "global", y `layout_margin<Lado>`, con Lado uno de Bottom, Left, Right o Top). ¿Para qué sirve? ¿Crees que funciona bien? Prueba a cambiar a la vez Gravity.
- Manipula las propiedades Gravity y Padding (asegúrate de que tienes `layout_height` y `layout_width` en `fill_parent`). Estas propiedades son ya del control, no del layout. ¿Para qué sirven? ¿En qué se diferencia `layout_gravity` de Margin?
- Haz pruebas similares con gravity y Padding, pero dejando `layout_*` en `wrap_content`. ¿Sirven para algo? ¿Qué hacen?

Las dimensiones se especifican en "dp" (density-independent pixel). Un dp es lo mismo que un pixel en una pantalla de 160 dpi (dots per inch). Android hace la conversión a píxeles en pantalla sobre la marcha en función del hardware en el que se ejecute.

Práctica 2.11: Etiquetas

Hasta ahora hemos utilizado únicamente botones en nuestras actividades. Vamos a experimentar con el funcionamiento de las etiquetas.

Son un tipo de control muy sencillo e incluso podemos considerar que ya sabemos usarlas: la clase `Button` hereda de `TextView` (las etiquetas). Además, el asistente de proyectos del ADT nos crea siempre una actividad con una única etiqueta con el texto Hello World!

- Crea un nuevo proyecto, de nombre Etiqueta. Ajusta el resto de valores como de costumbre.
- Elimina el texto asociado al XML del recurso por:

```
<TextView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/textView1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
```

```
android:gravity="center"
android:text="@string/hello_world" />
```

- El texto de la etiqueta se establece con el atributo `android:text`, que puede acceder a un recurso de tipo cadena al igual que con los botones.
- Haz las mismas pruebas sobre ella que las que hiciste con el botón. Ten en cuenta que con las etiquetas es más complicado hacerse una idea de qué está pasando, porque son "transparentes". Cuando se diseña un layout, a menudo es interesante poner fondos temporales a los controles transparentes, para tener una visión más clara de lo que está ocurriendo. Para eso, puedes poner en el atributo `android:background` un color con el formato `#RRGGBB` (por ejemplo, `#FF0000` para rojo).

Práctica 2.12: Cuadros de texto

Los cuadros de texto son controles donde el usuario puede introducir texto. Son proporcionados por la clase `EditText`, que es similar en dificultad de uso al resto (de nuevo, `EditText` hereda de `TextView`).

- Haz un nuevo proyecto y en el layout establece:

```
<EditText
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/editText"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:inputType="text"
    android:text="@string/escribeAlgo" />
```

- Hay un atributo adicional que no teníamos antes llamado `inputType`. Sirve para especificar el tipo de entrada. Especificamos `text` si queremos un texto genérico. Podemos configurar el control para que reciba un número, una contraseña, un nombre, una fecha, y un largo etcétera.
- Crea un nuevo recurso de tipo cadena con el identificador `escribeAlgo` y el texto `Escribe algo`.
- Haz las mismas pruebas sobre ella que las que hiciste antes.

Práctica 2.13: Linear Layout

Hasta ahora sólo hemos puesto un control en nuestra ventana. Si intentáramos añadir más de uno, obtendríamos un error, porque una ventana sólo puede tener un control raíz. Es necesario añadir contenedores, que hospedarán a otros controles o a otros contenedores. Construiremos así una jerarquía de controles que crean la composición final. Los controles que hemos visto (etiquetas, botones, y cuadros de edición) son "hojas" en ese árbol, y son subclases de `View`. Los contenedores agrupan controles, son subclases de `ViewGroup`, e implementan una política de organización de sus controles hijo en el espacio del que disponen, a la que se le denomina `layout`.

En esta práctica vamos a ver el funcionamiento de la clase `LinearLayout`, que organiza sus controles `_en línea_`, ya sea horizontal o verticalmente.

- Crea un nuevo proyecto de prueba, llámalo `LinearLayout` y deja el resto de valores con los predefinidos en el asistente.
- Abre el fichero del layout de la actividad principal, y sustitúyelo por el siguiente texto:

```

<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#E0E0E0"
    android:orientation="vertical" >

    <!-- Controles que contiene. Dos etiquetas -->
    <TextView
        android:id="@+id/text1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#80FF80"
        android:text="@string/etiqueta1" />
    <TextView
        android:id="@+id/text2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#FF8080"
        android:text="@string/etiqueta2" />
</LinearLayout>

```

Se ha puesto intencionadamente un color de fondo a cada etiqueta y al propio layout para que las pruebas que realicemos queden más claras.

- Define las cadenas etiqueta1 y etiqueta2 en el fichero de recursos, usando como texto Etiqueta 1 y Etiqueta 2.
- Ve a la vista Graphical layout y comprueba el resultado. Lanza la aplicación en un AVD para verlo también.
- Observa la estructura en árbol tanto en el XML como en la relación de los contenedores y controles finales que tienes disponible en la ventana Outline. Dentro del LinearLayout podríamos haber incluido otro LinearLayout.
- Selecciona en la ventana outline el LinearLayout.
 - ✓ Modifica las propiedades layout_width y layout_height estableciéndolas al valor wrap_content y comprueba el efecto. Vuelve a colocarlo en fill_content.
 - ✓ Modifica la orientación a horizontal y comprueba el efecto. ¿Qué ha ocurrido?
 - ✓ Modifica el atributo layout_width a wrap_content para la primera etiqueta. ¿Qué ocurre?
 - ✓ Vuelve a poner el LinearLayout en vertical. ¿Qué sucede?
 - ✓ Restaura a fill_parent el layout_width de la etiqueta 1.
 - ✓ Cambia el padding del LinearLayout a 10dp. ¿Qué ocurre?
 - ✓ Deshaz el último cambio. Modifica el layout_margin del LinearLayout a 10dp. ¿En qué se diferencia del resultado anterior?
 - ✓ Deshaz el último cambio. Modifica el layout_margin de la etiqueta 1 a 10dp. ¿En qué se diferencia de los anteriores?
 - ✓ Deshaz el último cambio. Modifica la propiedad gravity del LinearLayout a bottom. ¿Qué ocurre?
 - ✓ Vuelve a poner el LinearLayout con orientación horizontal, y la etiqueta 1 con layout_width en wrap_content.
 - ✓ Pon también wrap_content en layout_width en la etiqueta 2. Ambas quedarán juntas, ocupando el menor espacio posible.

- ✓ El espacio sobrante podemos repartirlo entre los controles contenidos por el layout usando la propiedad `layout_weight`. El `LinearLayout` suma el valor de esa propiedad de todos los elementos que contiene, y reparte el espacio sobrante equitativamente según esos pesos.
- ✓ Cambia el `layout_width` del `LinearLayout` a `wrap_content` y comprueba el efecto.

Práctica 2.14: ¡Adivina mi número!

Vamos a hacer un programa completo, en el que diseñaremos un interfaz en el que tendremos que capturar eventos y manipular la salida. En concreto, se trata del conocido juego "adivina mi número". El programa escoge un número aleatorio entre 1 y 100, y el usuario debe adivinar cuál es con sucesivos intentos. Tras cada uno, el programa le indicará si el número pensado es mayor o menor.

El layout debería ser similar al de la figura.

Cuando el usuario introduce un número y pulsa el botón Probar, el programa compara el número con el introducido y sustituye el texto superior por ¿NN? ¡Uy! El número que he pensado es mayor (o menor), donde NN es el número escrito por el usuario. Si acierta, deberá mostrar !!Has acertado!!.

Todo el texto debe aparecer en el fichero `strings.xml` para poder ser localizado. Algunas pistas:



- Para no construir manualmente el texto mostrado ante un error, puedes poner como recurso la cadena `¿ %1$d? ¡Uy! El número que he pensado es mayor.` Luego, se construye el texto final buscando la plantilla en los recursos, y formateándola sustituyendo el `%1$d` por el número introducido por el usuario:

```
String format = getResources().getString(R.string.<id>);
String cadFinal = String.format(format, numUsuario);
```

- Para generar un número aleatorio puedes utilizar la clase estándar `java.util.Random()`. Crea una instancia de esa clase en el `onCreate()`, guárdala en un atributo llamado, por ejemplo, `dado`, y utiliza:

```
numeroAAdivinar = dado.nextInt(100) + 1;
```

No obstante, durante las pruebas haz que el número a adivinar sea siempre el mismo para conocerlo y estar seguro de que el programa funciona.

- En el método llamado desde el evento de pulsación del botón, el objeto View del parámetro será el botón no el cuadro de texto. Necesitarás usar `findViewById(...)` para conseguirlo.
- Para extraer el texto escrito en un EditText se usa:

```
String numUsuarioTxt = miEditText.getText().toString();
```

- Para convertir una cadena a número puedes usar:

```
int numUsuario = Integer.parseInt(numUsuarioTxt);
```

Además, aquí van algunas ideas, algo más avanzadas, para mejorar el programa:

- A nivel de usabilidad, es molesto tener que introducir el número y luego pulsar el botón. Puedes eliminar el botón, y capturar el evento de "pulsación de intro" en el cuadro de texto, aunque no es muy sencillo:

```
et.setOnKeyListener(new android.view.View.OnKeyListener() {
    public boolean onKey(View v,
                        int keyCode,
                        android.view.KeyEvent event) {
        // Han pulsado (o soltado) una tecla...
        if ((event.getAction() ==
            android.view.KeyEvent.ACTION_DOWN) &&
            (keyCode ==
            android.view.KeyEvent.KEYCODE_ENTER)) {
            // Ha sido una pulsación de "intro"
            // PENDIENTE! HACER ALGO!!
            return true;
        }
        else
            return false;
    } // onKey
});
```

- Puedes mejorar el programa para que cuando acabe la partida, se deshabilite (o incluso oculte) el cuadro de texto (y el botón, si sigues con él), y aparezca un nuevo botón que pueda ser pulsado para volver a jugar. Para eso, es interesante saber que los controles tienen un atributo `android:visibility` que puede tomar (en el XML) los siguientes valores:
 - ✓ **visible**: el control es visible y aparece en su posición en función de la organización del interfaz. Es el valor predefinido.
 - ✓ **invisible**: el control no es visible pero sigue ocupando el espacio que le correspondería si lo estuviera.
 - ✓ **gone**: el control no es visible y no ocupa espacio. Si se muestra, desplazará otros controles para ajustar la disposición global.

Desde código podemos usar el método `setVisibility()`, y pasar por parámetro una de las constantes `VISIBLE`, `INVISIBLE` o `GONE`, definidas en la clase `android.View`.

- ✓ Por último, ¿qué tal añadir una etiqueta que diga cuantos intentos llevas?

Notas.

Podemos utilizar Mensajes como:

```
<string name="mensaje">Has pulsado %1$d <!--%d-->veces</string>
```

Luego cargaremos el mensaje y lo utilizaremos.

```
public void pulsarBoton(View v){
    ++numVeces;
    _boton=(Button) findViewById(R.id.boton);
    // _boton.setText("has pulsado "+numVeces+" veces");
    String text = getString(R.string.mensaje, numVeces);
    _boton.setText(text);
}
```

- 🔗 Enlace para personalizar el botón, textview, editext, etc.

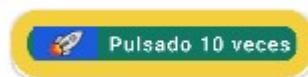
<https://es.acervolima.com/como-crear-botones-personalizados-en-android-con-diferentes-formas-y-colores/>

Ejercicios.

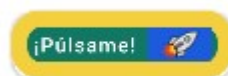
1.- Crea un botón que cambie el color del fondo de la pantalla cada vez que se pulse, el color tiene que ser aleatorio.

2.- Personaliza el botón.

- ✓ El texto del botón por defecto está en mayúsculas con: `android:textAllCaps="false"` lo desactivamos.
- ✓ Para desactivar el formato del botón tendremos que poner: `app:backgroundTint="@null"` Solamente funciona a partir de API 21.
- ✓ Podemos añadir alguna imagen al botón (a la izquierda, derecha, arriba o abajo):
 - `android:drawableLeft="@drawable/cohete"`



- `android:drawableRight="@drawable/cohete"`
- `android:drawableEnd="@drawable/cohete"`



- `android:drawableTop="@drawable/cohete"`



o `android:drawableBottom="@drawable/cohete"`

