Unidad 1. Tecnologías para aplicaciones en dispositivos móviles

PONTE A PRUEBA 1

Conociendo nuestro dispositivo (II)

Vamos a seguir conociendo nuestro dispositivo; ahora es el turno de los sensores. ¿Conoces todos los sensores que contiene tu smartphone?

Investiga sobre los sensores de tu smartphone y busca aplicaciones que podrían mostrar su funcionamiento.

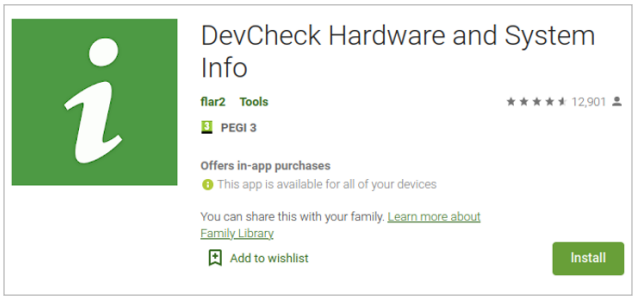
Claves de resolución

Existen apps, como *DevCheck Hardware and System Info*, o *Sensors Test*, que nos ofrecen una comprobación de los sensores de nuestro equipo.

También existen sitios web que nos permiten conocer las características y los sensores de diferentes dispositivos, como **Kimovil**.

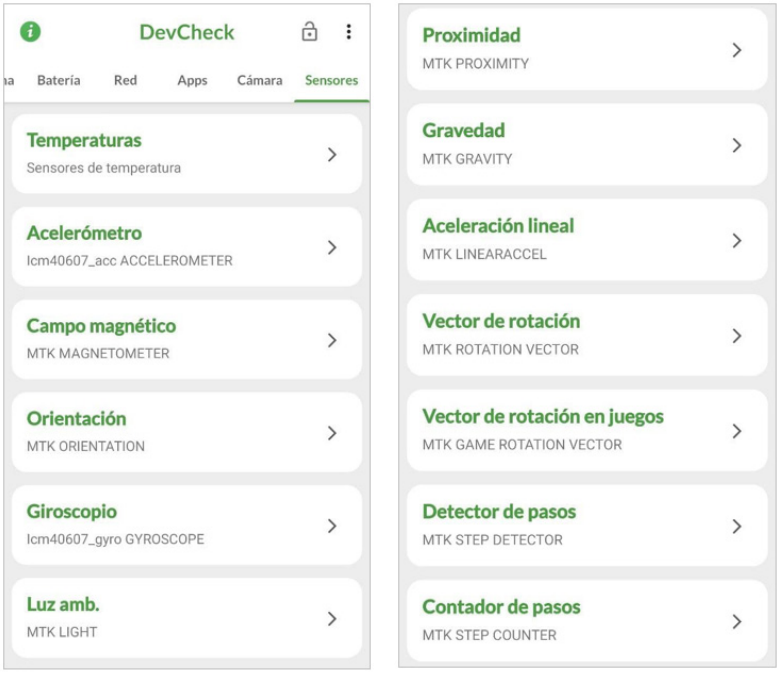
SOLUCIÓN

Para la solución de este caso, vamos a instalar la aplicación *DevCheck Hardware and System Info* desde la Play Store sobre el Redmi Note 8 que utilizamos en el caso anterior. Cada alumno podrá usar su propio dispositivo para comprobar sus sensores.



Esta aplicación también nos permite conocer más especificaciones del hardware de nuestro smartphone, como la densidad de la pantalla, la tasa de refresco y muchos detalles sobre las cámaras.

Pero para el caso que nos interesa nos centramos en la opción de los sensores, ubicada al final del menú. En esta se listan los posibles sensores de nuestro dispositivo, entre los que podemos encontrar de temperatura, acelerómetro y giroscopio, campo magnético, orientación, gravedad, detector de luz ambiental, proximidad, aceleración lineal, rotación de varios tipos y contadores de pasos.



En el caso que se desee trabajar más el tema de los sensores, puede ser de utilidad el artículo **Diez cosas que puedes medir usando solo sensores de tu móvil**, en el que se comentan algunos sensores interesantes y varias aplicaciones para probarlos.

PONTE A PRUEBA 2

Probando nuestro Device Manager.

Si todo ha ido bien en la instalación de nuestro dispositivo virtual, es hora de ponerlo en marcha. Para ello, tal y como hemos visto en el CPE anterior, es posible lanzarlo tanto desde el Device Manager como desde la terminal.

Inicia tu dispositivo virtual y comprueba que funciona correctamente, así como la versión del sistema que tiene instalada. Personaliza también tu dispositivo configurando, por ejemplo, tu idioma preferido.

Claves de resolución

Recuerda que puedes comprobar la versión del sistema a través de la opción *Ajustes*.

SOLUCIÓN

Podemos lanzar el emulador desde el Device Manager del propio Android Studio o desde la terminal. Desde esta última, deberíamos ubicarnos en el directorio donde tuviésemos instalado el emulador, dentro del Sdk, y ejecutar el comando:

```
$ ./emulator -list-avds
```

Se muestra la lista de dispositivos virtuales disponibles. En nuestro caso, si solamente tenemos disponible el Pixel 3, sería:

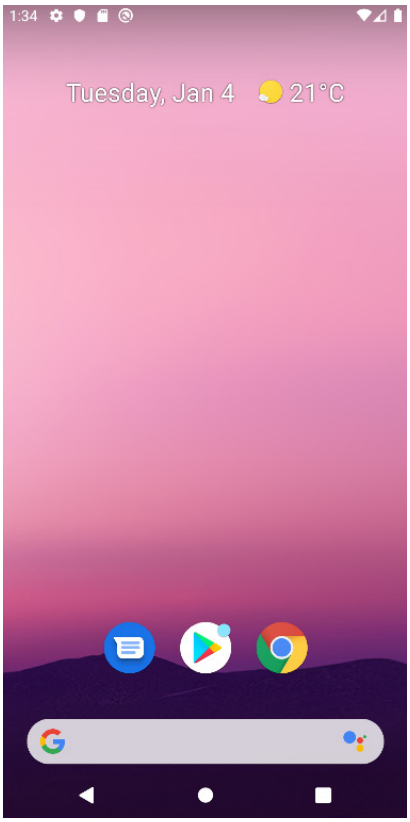
```
Pixel_3_API_29
```

Para lanzar este dispositivo, haríamos:

```
$ emulator @Pixel_3_API_29 -accel on
```

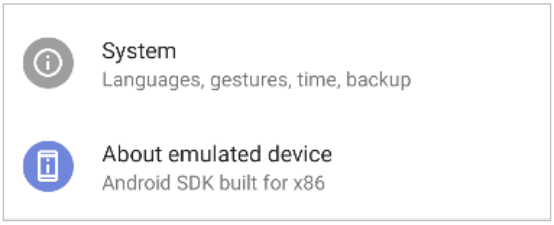
De todos modos, seguramente nos resulte más cómodo lanzar el emulador desde el Device Manager.

Sea como sea, veremos cómo se inicia el terminal con la animación de arranque del sistema y muestra la pantalla de inicio:

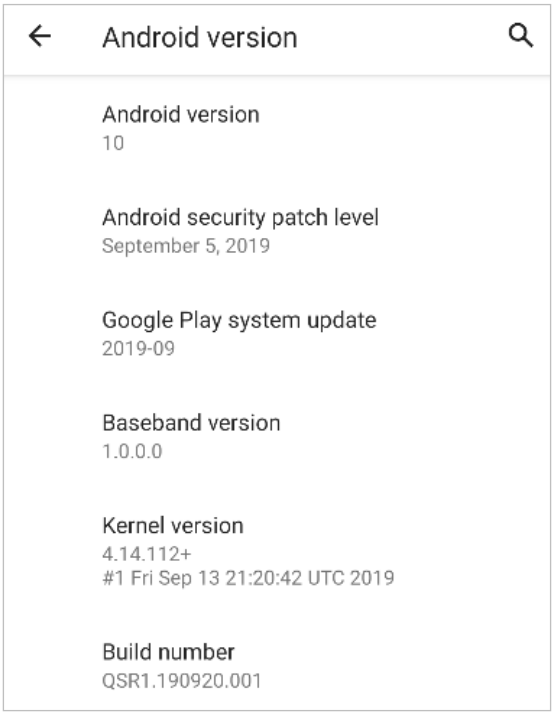


Iremos a ajustes del menú para comprobar la versión del sistema. Para acceder al menú de aplicaciones, deberemos imitar el gesto de arrastrar el dedo en la pantalla hacia arriba, pulsando con el botón sobre ella y arrastrando hacia arriba.

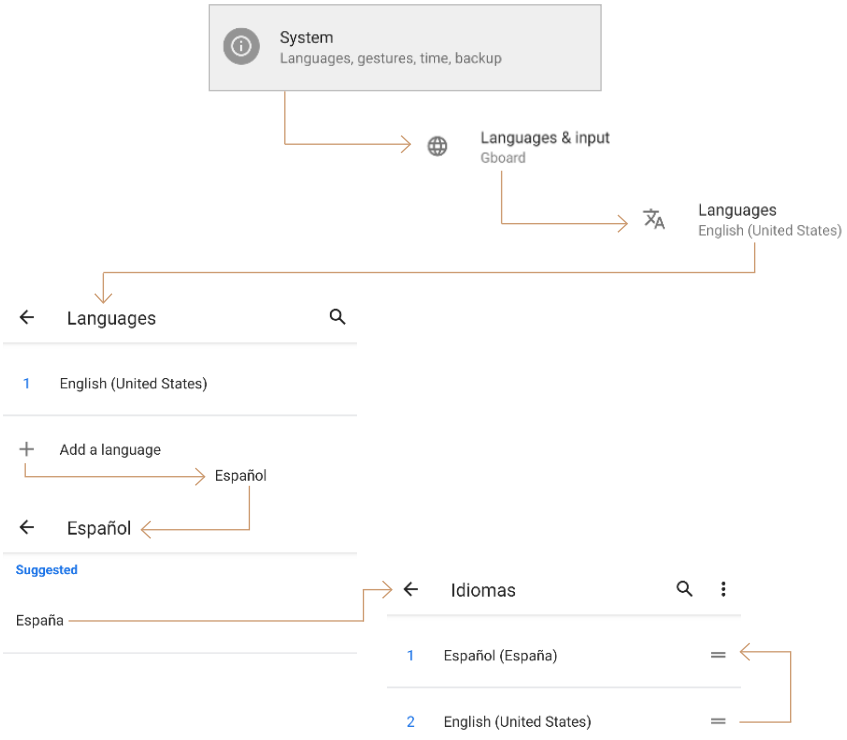
Accedemos entonces a *Settings* y nos desplazamos (haciendo clic y arrastrando con el ratón) hacia las últimas opciones, *System* y *About emulated device*:



Desde esta última podemos acceder a opciones diversas sobre el dispositivo, entre las que encontramos la versión de Android. Haciendo clic en ella, obtenemos más opciones:



Para modificar el idioma, podemos configurarlo en la penúltima opción de los ajustes (System) mediante la opción *Language & Input*. Desde ahí, seleccionamos la opción de idiomas y añadimos un idioma nuevo. Seleccionamos, por ejemplo, español de España y lo arrastramos hasta la primera posición en la lista de idiomas disponibles.



PONTE A PRUEBA 3

Análisis del Hola Mundo

A partir del proyecto creado en el caso práctico anterior, examina su estructura y código para contestar las preguntas siguientes:

1. ¿Para qué versión del SDK se ha creado la aplicación?
2. ¿Dónde se cargan las librerías de compatibilidad de Jetpack?
3. ¿Qué es y dónde podemos encontrar la cadena *com.mgh.pmdm.holamundo*?
4. ¿Cuál es el icono principal de la aplicación y qué etiqueta tendrá en el menú de inicio?
5. ¿Qué componentes principales define la aplicación? ¿Qué características tienen?
6. ¿Dónde se encuentran definidos los recursos de diseño de la interfaz?
7. Examina el código de la clase MainActivity que implementa la Actividad. ¿Qué crees que hace la línea `setContentView(R.layout.activity_main)`?

Claves de resolución

- Gran parte de la información se puede encontrar en los scripts de Gradle y en el archivo de manifiesto.
- Recuerda que tienes dos perspectivas diferentes para explorar el proyecto, la vista de ficheros (Project Files) y la vista de Android.

SOLUCIÓN

1. ¿Para qué versión del SDK se ha creado la aplicación?

Para la versión 31. Esta información la podemos encontrar en el fichero app/build.gradle, dentro de *Android > compileSdk*.

2. ¿Dónde se cargan las librerías de compatibilidad de Jetpack?

Las encontramos como dependencias, en el apartado *dependencies* del fichero build.gradle de la aplicación. Concretamente en esta línea:

```
implementation 'androidx.appcompat:appcompat:1.4.0'
```

3. ¿Qué es y dónde podemos encontrar la cadena *com.mgh.pmdm.holamundo*?

La encontramos en las siguientes ubicaciones:

- En el fichero build.gradle de la aplicación, en el campo applicationId, representando el id único de la aplicación.
- En el fichero AndroidManifest.xml, definiendo el nombre del paquete de la aplicación.
- En los diferentes ficheros fuente (en este caso MainActivity.tk), como el nombre del paquete al que corresponden las clases.

En este apartado es interesante puntualizar que esta cadena no tiene que ser la misma en el id de la aplicación (que es usado internamente por Android y Google Play) del nombre del paquete definido en el manifiesto, aunque lo habitual —y menos problemático— es usar la misma.

4. ¿Cuál es el icono principal de la aplicación y qué etiqueta tendrá en el menú de inicio?

Esta información se encuentra dentro de la etiqueta application en el fichero *AndroidManifest.xml*, concretamente, en los atributos android:icon (@mipmap/ic_launcher) y android:label (HolaMundo). El icono ic_launcher se encontrará dentro de la carpeta res/mipmap/ic_launcher. Si examinamos esta carpeta, veremos que existen distintos ficheros para diferentes densidades de pantalla.

5. ¿Qué componentes principales define la aplicación? ¿Qué características tienen?

Dentro de application (en el fichero AndroidManifest.xml), podemos ver que únicamente se define un componente de tipo Actividad (*activity*). Por tanto, podemos decir que:

- Su nombre es com.mgh.pmdm.holamundo.MainActivity (concatenando el nombre del Package y el atributo android:name). Además, observamos que el nombre coincide con el nombre de la única clase que hay en la actividad (com/mgh/pmdm/holamundo/MainActivity.tk).
- Esta actividad, además, puede recibir *Intents*, concretamente la acción android.intent.action.MAIN (es la actividad principal de nuestra aplicación), y aparecerá en el menú de aplicaciones ya que tiene la categoría de LAUNCHER).

6. ¿Dónde se encuentran definidos los recursos de diseño de la interfaz?

Encontramos el recurso en la carpeta *res/layout/activity_main.xml*, como un fichero XML.

7. Examina el código de la clase MainActivity que implementa la Actividad. ¿Qué crees que hace la línea `setContentView(R.layout.activity_main)`?

Esta es quizá la parte más complicada, pero también la más interesante, ya que enlaza varios conceptos de los vistos en el apartado.

Como su nombre indica (y aún si conocer Kotlin), la traducción directa sería 'establece el contenido de la vista', es decir, la actividad en sí.

Sin embargo, lo más interesante es que aquí entra en acción la clase R, que hace referencia a los recursos, y vemos la forma de utilizarla. Con *R.layout* estamos accediendo a la carpeta *layout* dentro de los recursos, y, dentro de ella, a su vez a un recurso llamado *activity_main*, que no es otro que el definido en el fichero layout/activity_main.xml, aunque no se indica la extensión.

Así pues, esta línea establece como diseño para esta actividad el XML indicado dentro de la carpeta de recursos.

Más adelante veremos que esta vinculación de vistas y actividades se suele realizar mediante lo que se conoce como View Binding, que nos facilitará el acceso a las diferentes vistas de la interfaz.

PONTE A PRUEBA 4

Más figuras

Siguiendo con el Caso práctico 4, crea ahora las clases y funcionalidades correspondientes para:

- Crear un rectángulo de base 3 y altura 5, de color azul.
- Crear un cuadrado de lado 4, de color verde.
- Crear un triángulo base 2 y altura 5, de color naranja.
- Crear un círculo de radio 7, de color rojo.
- Crear una elipse de radios 5 y 6, de color amarillo.

Y que se muestre el área de cada objeto y su color, del mismo modo que hemos hecho en el caso anterior.

Claves de resolución

- Observa bien cómo se ha implementado la clase Rectángulo, ya que te puede ser de gran ayuda para resolver este caso.
- Recuerda el cálculo del área de las diferentes figuras geométricas:
  - Cuadrado: lado x lado
  - Triángulo: (base x altura)/2
  - Círculo: Pi x R2
  - Elipse: Pi x Radio1 x Radio 2.

SOLUCIÓN

Para abordar este ejercicio podemos tomar como plantilla la clase Rectángulo definida en el caso práctico y adaptarla a las peculiaridades de las nuevas clases. Aparte del método para calcular el área, también deberemos tener en cuenta los parámetros con que se inicializa en el constructor de cada subclase.

El código resultante es el siguiente:

```
interface Superficie{
    fun mostrarArea()
}

open class Figura(val color:String){
    fun mostrarColor(){
        println("Color:"+color)
    }
}

class Rectangulo(val base:Int, val altura:Int, color:String):Figura(color),
Superficie {

    override fun mostrarArea(){
        println("Area del rectángulo: "+base*altura)
    }
}
```

```
class Cuadrado(val lado:Int, color:String):Figura(color), Superficie {

    override fun mostrarArea(){
        println("Area dek cyadradi: "+lado*lado)
    }
}

class Triangulo(val base:Int, val altura:Int, color:String):Figura(color),
Superficie {

    override fun mostrarArea(){
        println("Area del triángulo: "+(base*altura)/2)
    }
}

class Circulo(val radio:Int, color:String):Figura(color), Superficie {

    val pi=3.1416

    override fun mostrarArea(){
        println("Area del círculo: "+pi*radio*radio)
    }
}

class Elipse(val radio1:Int, val radio2:Int, color:String):Figura(color),
Superficie {

    val pi=3.1416

    override fun mostrarArea(){
        println("Area de la elipse: "+pi*radio1*radio2)
    }
}

fun main(){
    var fig1=Rectangulo(3, 5, "azul")
    fig1.mostrarArea()
    fig1.mostrarColor()

    var fig2=Cuadrado(4, "verde")
    fig2.mostrarArea()
    fig2.mostrarColor()

    var fig3=Triangulo(2, 5, "naranja")
    fig3.mostrarArea()
    fig3.mostrarColor()

    var fig4=Circulo(7, "rojo")
```

```
    fig4.mostrarArea()
    fig4.mostrarColor()

    var fig5=Elipse(5, 6, "amarillo")
    fig5.mostrarArea()
    fig5.mostrarColor()
}
```

PONTE A PRUEBA 5

Sumando y restando

Se propone un nuevo proyecto, U1PP5Contador (fichero U1PP5Contador.zip), con un nuevo diseño del proyecto en el que ahora aparecen los botones + y - para incrementar y decrementar el contador.

Se pide incorporar la funcionalidad para que el contador incremente cada vez que se haga clic en el botón «+» y decremente cada vez que se haga clic en el botón «-».

Además, deberemos mantener el estado de la actividad a lo largo de su ciclo de vida.

Claves de resolución

- Los pasos a seguir serán los mismos que hemos visto en este apartado.
- Examina el Layout proporcionado en el proyecto para averiguar los identificadores de los botones.

SOLUCIÓN

Si examinamos el diseño, veremos que el botón de sumar sigue teniendo el identificador *BtAdd*, y que el identificador del botón de resta es *BtRest*.

Así pues, al código que teníamos del proyecto deberemos añadirle el código para capturar el evento *onClick* sobre el botón de restar e incorporar el código correspondiente:

```
btRest.setOnClickListener {
            contador--
            textViewContador.setText(contador.toString())
        }
```

Disponéis del código completo en el proyecto U1PP5Contador_Solucion.zip.