Introducción

Lo adivinaste. Tampoco te voy a explicar nada de **Kotlin**. No necesitas saber nada. En el fondo, Kotlin se basa en la Java Virtual Machine. De hecho, Android Studio puede traducir **Java ←→ Kotlin** él solito sin mucho problema. Desde el punto de vista del desarrollador, Kotlin es como Java, pero *sin el código redundante* que tiene a veces Java. Vamos a aprender Kotlin sobre la marcha. **El objetivo** de esta práctica es implementar un sencillo *Hola Mundo* en Android e instalarlo en un dispositivo móvil (o emularlo). Lo mismo que antes.

Se pide: Una App en Android que disponga de un **Button** y un **TextView**. El **Button** tiene que estar etiquetado como "Ok". El **TextView** mostrará el texto "Pulsa el botón". Cuando el usuario presiona dicho botón, el **TextView** mostrará el texto "Hola Mundo". Si el usuario presione el botón de nuevo, el **TextView** mostrará otra vez "Pulsa el botón".

SOLUC: P0 - ButtonAndTextViewKotlin.zip

Primer paso – Nuevo proyecto

No hay nada diferente respecto a la práctica en Java, salvo decirle que vas a usar Kotlin en vez de Java.

Segundo paso – La interfaz

No hay nada diferente respecto a la práctica en Java. Puedes limitarte a copiar los xml del layout y los textos de la práctica de Java.

Tercer paso – La lógica de negocio

Vamos ahora a la clase **MainActivity.kt**. En esta clase se describe el comportamiento cada uno de los elementos de la parte del diseño. Por defecto, el Android Studio nos da este código:

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
```

Este código es similar al de Java. La función **onCreate** se lanza siempre ella sola cuando se inicia la App. No es un constructor de una clase normal de Java, pero se usa como si lo fuera. Las dos líneas que tiene siempre tienen que dejarse como están al principio del método, nosotros añadiremos nuestro código debajo. Si te fijas, las referencias a los recursos se hacen de la misma forma que en Java.

Para poder utilizar el **Button** y el **TextView**, es necesario que Kotlin los referencie. Y de nuevo, se hace como en Java. Esta vez lo vamos a colocar en el **onCreate**, donde debería estar siempre tanto en java como en Kotlin:

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    val textView : TextView = findViewById(R.id.textView)
    val button : Button = findViewById(R.id.button)
```

En Kotlin, las variable se definen de forma diferente. Primero, se define la variable con **val** o **var**. Un **val** es una variable de solo lectura, cuyo valor no va a cambiar nunca. Equivale a poner final en Java. En cambio, si ponemos **var** estamos diciendo que podemos reasignar el valor de dicha variable. Por ejemplo, haciendo que button apunte a otro button. Te irás dando cuenta que el Android Studio es muy puntilloso y te salta warnings cada vez que una variable debería ser de un tipo o de otro. Los **dos puntos** sirven para decir el tipo de variable que es, en este caso, un TextView y un Button. Kotlin permite incluso prescindir del tipo de variable.

No nos queda más que indicar el evento al **Button** que hará cambiar el texto del **TextView**. No nos vamos a complicar la vida y vamos a hacerlo como Kotlin y Android Studio quiere que le asignemos eventos:

```
button.setOnClickListener {

    val textViewText = textView.text.toString()

    val textOk = getString(R.string.text_click)
    val textHello = getString(R.string.text_hello_world)

    textView.text = if (textViewText == textOk) textHello else textOk
}
```

No. No es nada complicado de entender. La variable **textViewText** no necesita que digamos su tipo porque Kotlin es capaz de deducirlo a partir del retorno del método toString(). Como devuelve un String, Kotlin asume que **textViewText** también es un String.

Seguimos. ¿Te acuerdas de los POJO de java? Eran clases con atributos, getters y setters. ¿Nunca te has preguntado por qué estamos obligados a poner los getters y setters? En Java tenía sentido. En Kotlin no. En Kotlin se asume que ya existen. Por tanto, si tu accedes a **textView.text** en realidad estás haciendo un **textView.getText**().

Por último, lo que vemos en el ejemplo es otra forma resumida de hacer un if-else. Kotlin quiere las cosas se hagan así. Lo que está haciendo esta línea es que, si los String son iguales o no, se hace un textView.setText() de una cosa u otra. Fíjate que no hay equals como en Java. O bueno lo hay, pero no lo ves. Esta estructura de if-else y asignación es muy habitual en Kotlin.

Por supuesto, puedes usar un if-else normal si lo prefieres…

Cuarto paso – Emulación

No hay nada diferente respecto a la práctica en Java.

Quinto paso – Instalando la app en tu móvil (Opcional)

No hay nada diferente respecto a la práctica en Java.