INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

INGENIERÍA EN SISTEMAS COMPUTACIONALES

MATERIA: APPLICATION DEVELOPMENT FOR MOBILE DEVICES

PROFESOR: CIFUENTES ALVAREZ ALEJANDRO SIGFRIDO

PRESENTA:

RAMIREZ BENITEZ BRAYAN

GRUPO: 3CM17

“TAREA – Kotlin Parte 1”

CIUDAD DE MEXICO A 3 DE MAYO DE 2022

Parte III

Conversión de cualquier archivo Java a Kotlin.

Una característica del plugin de Kotlin que es particularmente útil por su capacidad de convertir cualquier archivo fuente de Java a Kotlin, a la vez que mantiene una compatibilidad total con el tiempo de ejecución.

El poder ver exactamente cómo se traduciría cualquier archivo de Java a Kotlin puede ser útil si se escribe algo en Kotlin, siempre puede escribirlo en Java y luego usar esta característica para convertir ese código en Kotlin.

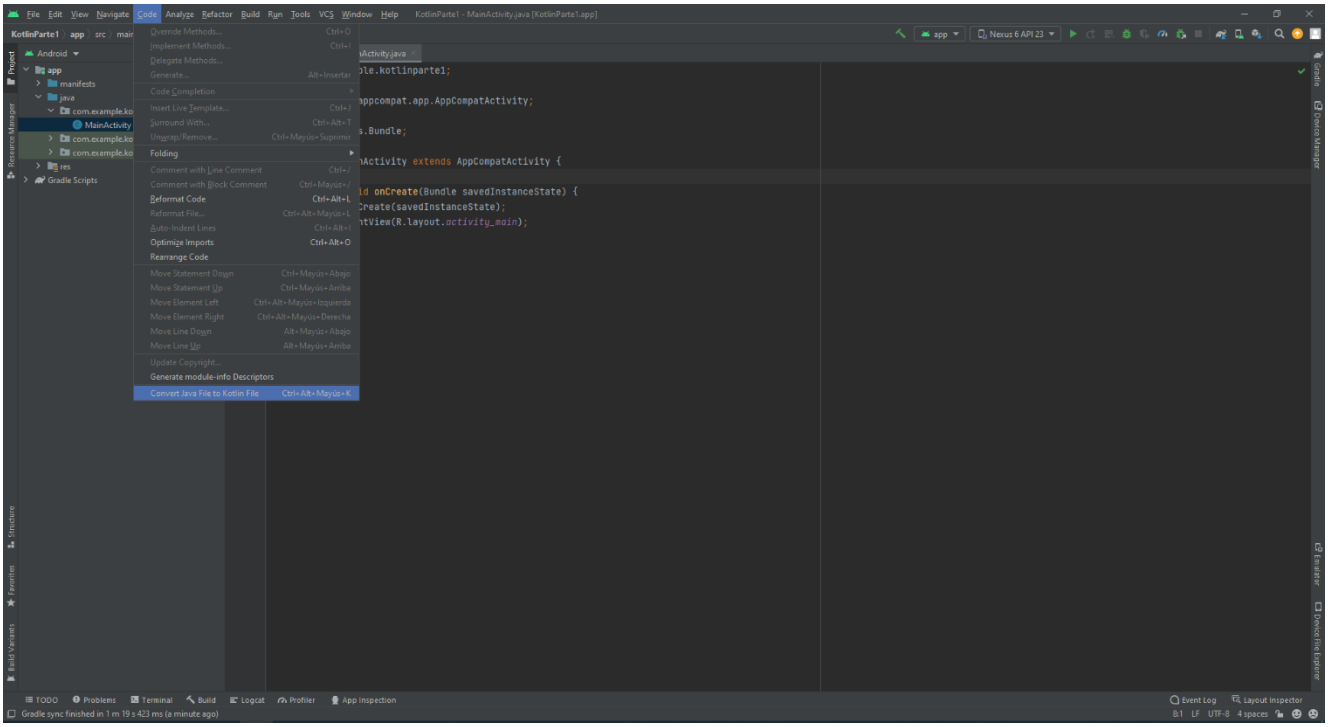
Por ejemplo, existen dos formas de invocar el archivo Convert Java file to Kotlin File del plugin de Kotlin, por lo que:

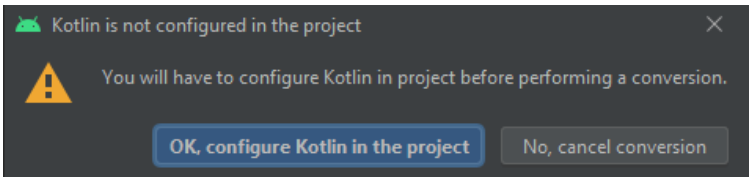
Seleccionar el archivo MainActivity y luego seleccionar Code en el menú de Android Studio, y enseguida seleccionar Convert Java File to Kotlin File.

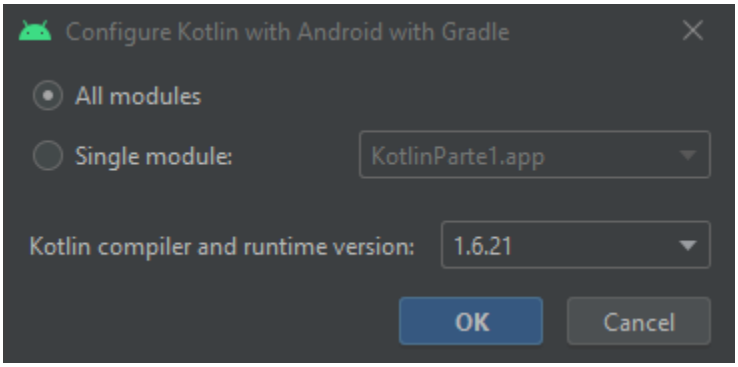
O seleccionar Help en la barra de menú de Android Studio, y luego seleccionar Find Action. En la ventana emergente siguiente, escribir Convert Java file to Kotlin file y luego seleccionar esta opción cuando se muestre.

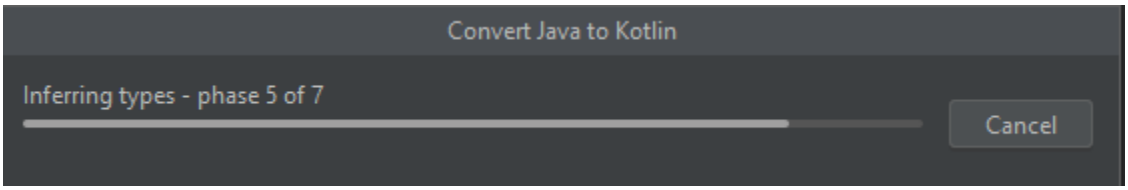
Hay que considerar que también se puede abrir la ventana emergente Find Action con un atajo de teclado; por ejemplo, si está en una Mac, presionar las teclas Command-Shift-A, y en Windows o Linux, presionar la tecla Control-Shift-A.

Hay que considerar que, dependiendo de la complejidad de su código, la conversión puede no ser siempre 100% precisa, por lo que siempre se debe verificar el código convertido para detectar errores.

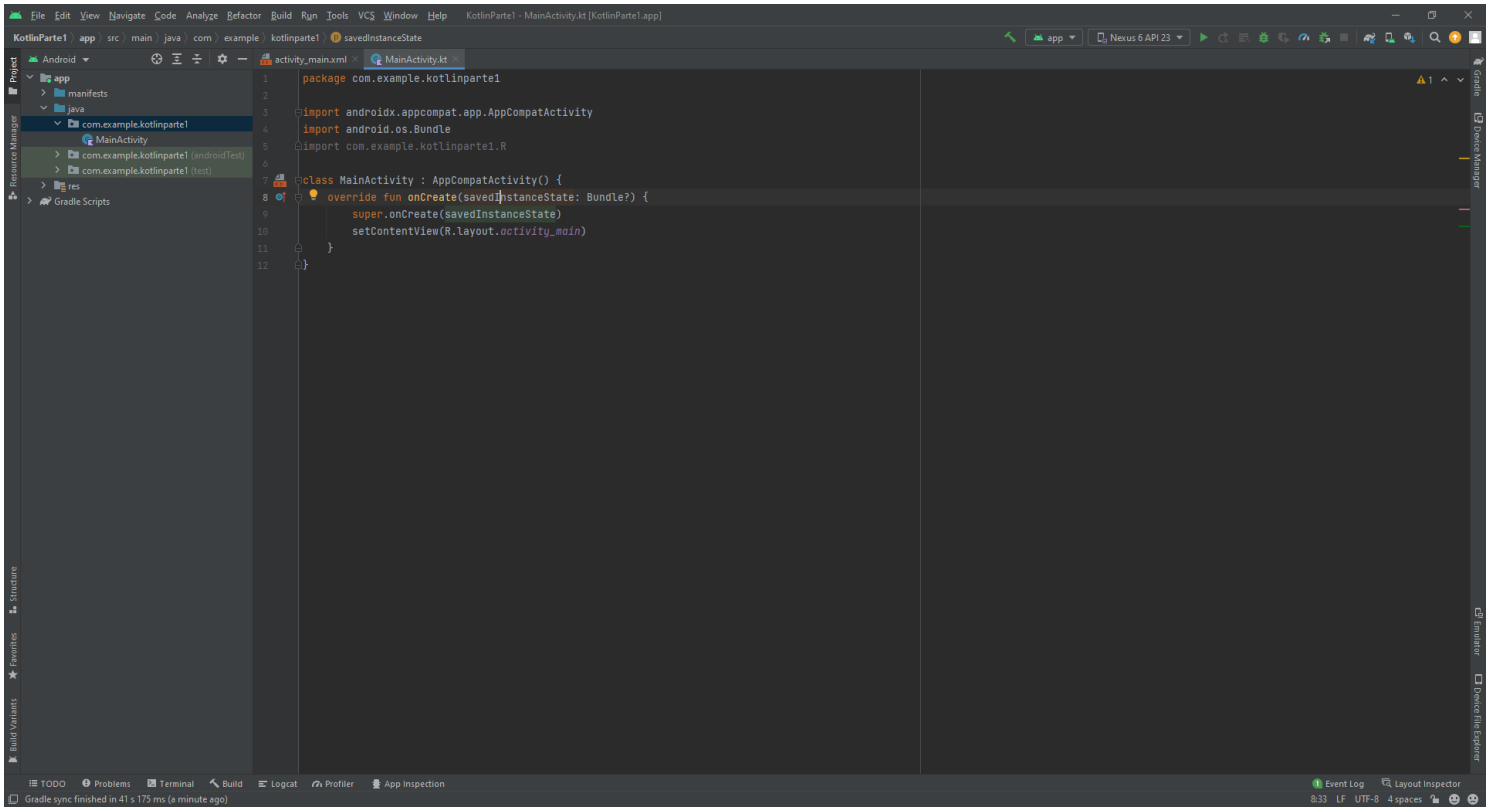








El archivo MainActivity recién convertido es el siguiente:



Parte IV

Revisión de la sintaxis de Kotlin.

En Kotlin, las clases se declaran usando la clase de palabra clave class, al igual que en Java. Sin embargo, en Kotlin, las clases (y los métodos) son públicos y finales por definición, por lo que se puede crear una clase simplemente escribiendo clase MainActivity.

Cuando se trata de extender una clase, se reemplaza el extends de Java con dos puntos, y luego se adjunta el nombre de la clase padre. Entonces, en la primera línea del archivo MainActivity.kt, se está creando una clase pública y final llamada MainActivity que extiende a AppCompatActivity: class MainActivity : AppCompatActivity(){

El equivalente en Java sería:

public class MainActivity extends AppCompatActivity{

Si desea sobrecargar una clase o método, se deberá declararlo explícitamente como abierto o abstracto.

En Kotlin, las funciones se definen utilizando la palabra clave fun, seguida del nombre de la función y los parámetros entre paréntesis. En Kotlin, el nombre de la función está antes de su tipo:

override fun onCreate(savedInstanceState: Bundle?) {

Esto es lo contrario en Java, donde el tipo está del nombre:

public void onCreate(Bundle savedInstanceState)

Tener cuenta que no se está especificando que este método sea final, ya que en Kotlin todos los métodos son final por definición. El resto de esta Activity es bastante similar a Java. Sin embargo, las siguientes líneas muestran otra característica clave de Kotlin:

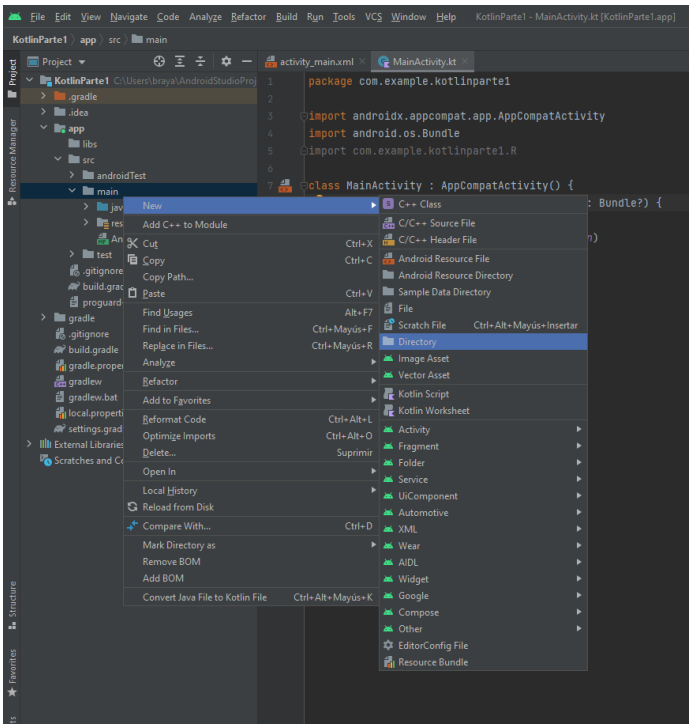
super.onCreate(savedInstanceState)

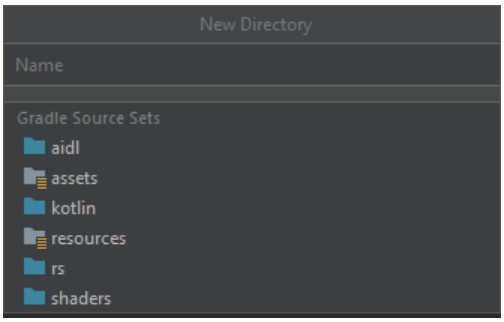
setContentView(R.layout.activity_main)

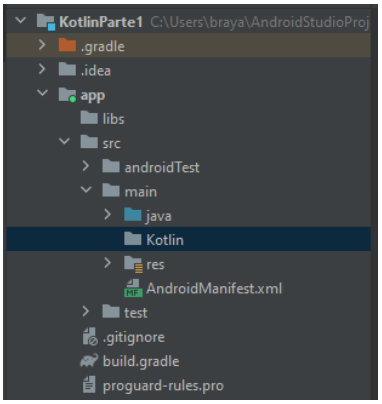
En Kotlin no se necesita que las líneas terminen con punto y coma, de ahí la ausencia de dos puntos en el fragmento anterior. Si se desea se pueden agregar dos puntos, pero el código será más limpio y fácil de leer sin ellos.

Como el plugin de Kotlin agrega una declaración src/main/kotlin al archivo build.gradle, se crea realmente esta carpeta. Este paso no es obligatorio, pero mantener sus archivos de Kotlin en una carpeta dedicada hará que el proyecto sea mucho más limpio.

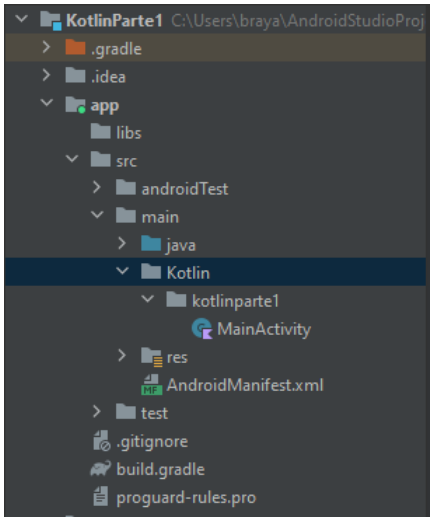
En el Project Explorer de Android Studio, digitar la tecla Control-Clic en el directorio Main del proyecto y seleccionar New en el menú que se muestra, seguido de Directory. Nombrar Kotlin a esta carpeta y luego clic en OK.







Una vez creado la carpeta dedicada a Kotlin, arrastrar el archivo MainActivity.kt dentro de él. Asegurarse de conservar el nombre del paquete existente del archivo MainActivity.kt para que el proyecto aún se ejecute.



Además, si solo se va a utilizar Kotlin en este proyecto, entonces se puede eliminar el directorio de Java, para no lidiar con directorios vacíos e innecesarios.

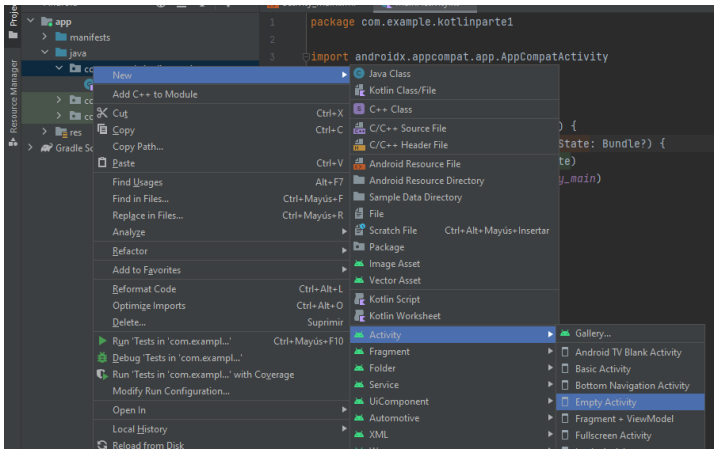
Dado que Kotlin compila en bytecode, una aplicación escrita en Kotlin se siente exactamente igual que una aplicación escrita en Java, así que, si se instala esta aplicación en un dispositivo Android o en un AVD compatible, debería parecer como si nada hubiera cambiado.

PARTE V.

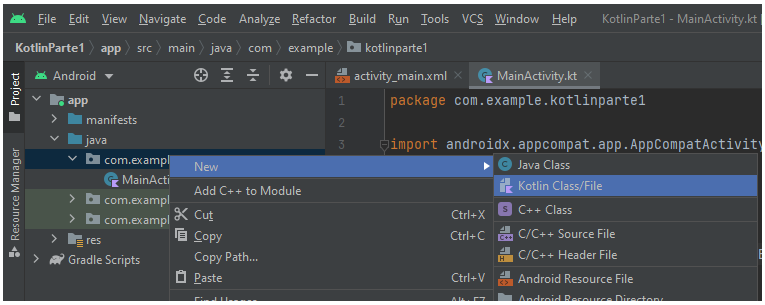
Creación de archivos adicionales de Kotlin.

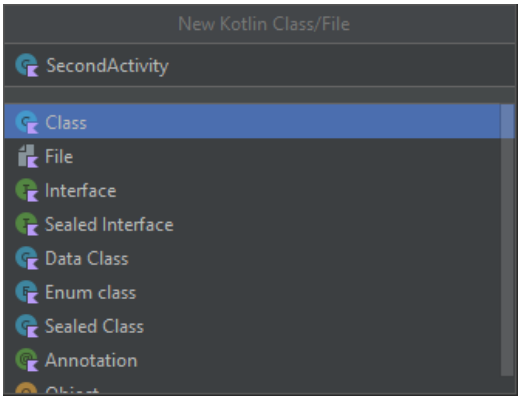
Si en el proyecto se continúa trabajando con Kotlin, seguramente se necesitará crear nuevos archivos de Kotlin en lugar de simplemente convertir los existentes de Java.

Para crear un archivo de Kotlin, presionar la tecla Control y clic en el directorio de la aplicación /src/main/kotlin y seleccionar New> Kotlin Activity.

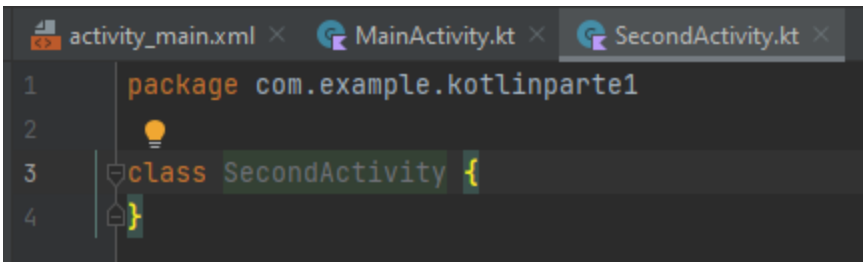


O también, en la ruta: \app\src\androidTest\java\com\example\escom\myapplication Asignar un nombre a la clase, por ejemplo, SecondActivity y seleccionar class en el menú desplegable:



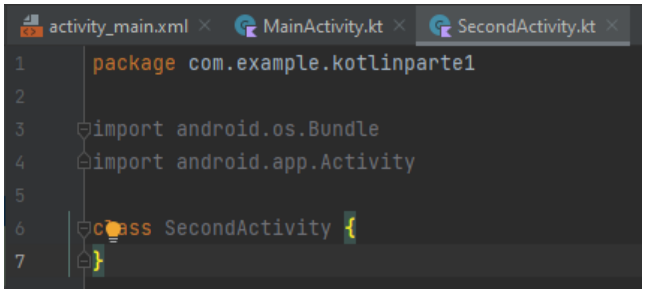


El código de la clase se encuentra vacío, como se indica enseguida:

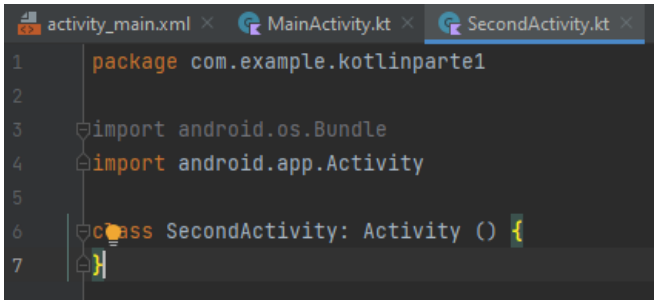


Para agregar alguna funcionalidad real, se completan algunos pasos. En primer lugar, agregar las declaraciones de importación. La única diferencia entre los enunciados import

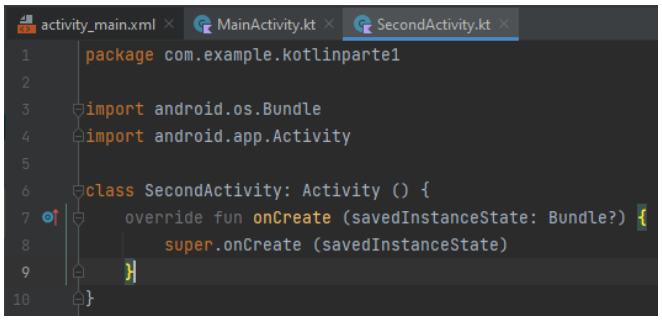
en Kotlin y los de Java es que no es necesario que termine cada línea con un punto y coma. Por ejemplo:



Enseguida, se especifica la herencia de la clase, utilizando el mismo formato del archivo MainActivity.kt:



Después, sobrecargar el método onCreate de la actividad:



Ahora se puede agregar la funcionalidad que se desee a esta actividad. Un último paso de configuración es declarar la actividad de Kotlin en el Manifest. Esto es igual que declarar una nueva actividad de Java, por ejemplo:

PARTE VI.

Extensiones de Android para Kotlin: Adiós a findViewById.

En Android, cada vez que se desee trabajar con cualquier View en una Actividad, se debe usar el método findViewById ara obtener una referencia a esa View. Esto hace de findViewById, uno de los códigos más importantes y frustrantes, sea una gran fuente de errores potenciales, y si se trabaja con múltiples elementos de la interfaz de usuario en la misma actividad, todos esos findViewByIds realmente pueden complicar el código.

La biblioteca Butter Knife elimina la necesidad de los findViewById, pero requiere que anote los campos para cada View, lo que puede generar errores y haría invertir mejor en otras áreas de su proyecto.

El plugin Kotlin Android Extensions evita utilizar findViewById y ofrece no tener que escribir algún código adicional o enviar un tiempo de ejecución adicional.

Se pueden usar las extensiones de Kotlin para importar las referencias de View en los archivos fuente. Aquí, el plugin de Kotlin creará un conjunto de "propiedades sintéticas" que permitirán trabajar con estas vistas como si fueran parte de la actividad; es decir, esto significa que ya no se tienen que usar findViewById para ubicar cada View antes de trabajar con ellos.

Para usar extensiones, se debe habilitar el plugin Kotlin Android Extensions en cada módulo. Abrir el archivo build.gradle a nivel de módulo y agregar lo siguiente:

```
apply plugin: 'kotlin-android-extensions'
implementation 'androidx.appcompat:appcompat:1.4.1'
implementation 'com.google.android.material:material:1.5.0'
implementation 'androidx.constraintlayout:constraintlayout:2.1.3'
testImplementation 'junit:junit:4.13.2'
androidTestImplementation 'androidx.test.ext:junit:1.1.3'
androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
implementation "androidx.core:core-ktx:+"
implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
```

Enseguida, sincronizar estos cambios haciendo clic en la ventana emergente Sync Now.

Se pueden importar las referencias a una solo View, utilizando el siguiente formato:

```
import android.os.Bundle
import android.app.Activity
import kotlinx.android.synthetic.main.*
```

Por ejemplo, si el archivo activity_main.xml contiene un TextView con el ID textView1, se debe importar la referencia a esta vista agregando lo siguiente a la actividad:

```
import android.app.Activity
import kotlinx.android.synthetic.main.*
import kotlinx.android.synthetic.main.activity_main.*
```

A continuación, se podrá acceder a textView1 dentro de esta actividad utilizando sólo su ID y sin findViewById a la vista.

Para las extensiones, se agrega un TextView al archivo activity_main.xml, importándolo al archivo MainActivity.kt y usando extensiones para establecer el texto de TextView programáticamente.

Primero se crea el TextView:

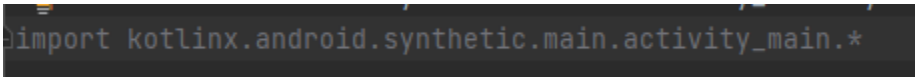
```xml
<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="10dp"
    android:paddingLeft="10dp"
    android:paddingRight="10dp"
    android:paddingTop="10dp"
    tools:context="com.example.kotlinparte1.MainActivity">

<TextView
    android:id="@+id/myTextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hola mundo!!"
    />
    </RelativeLayout>
```

Luego se importa el TextView en el MainActivity.kt, y se asigna su texto utilizando

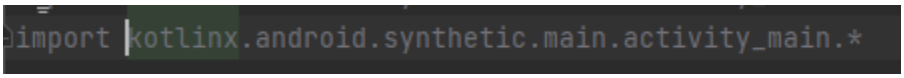
```kotlin
package com.example.kotlinparte1

import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import kotlinx.android.synthetic.main.activity_main.myTextView

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        myTextView.setText("Hello World")
    }
}
```

solamente su ID:

Observar que, si trabaja con varios widgets desde el mismo archivo de diseño, se puede importar todo el contenido de un archivo de diseño de una sola vez, usando lo siguiente:



Por ejemplo, si se querían importar todos los widgets del archivo activity_main.xml, se agregaría lo siguiente a la actividad:



Conclusiones:

Kotlin es un potente lenguaje de programación y una alternativa potencial a Java en el reemplazo como lenguaje estrella de Android. Cabe recalcar que Google hace de todo para que este lenguaje sea el oficial y el más utilizado por los desarrolladores de Android.