Asignatura: Application Development for Mobile Devices.
Tarea 32: Introducción a Kotlin.

Introducción

Kotlin es un lenguaje de programación estático de código abierto que admite la **programación funcional y orientada a objetos**. Posee variantes que se orientan a la JVM (Kotlin/JVM), JavaScript (Kotlin/JS) y el código nativo (Kotlin/Native).

Es 100% interoperable con Java y permite proyectos mixtos mezclando código Java con código Kotlin. Debido a que genera código de Java 6, es compatible con Android. Utilizando GWT (ver **Nota 1**) se puede también mezclar código Java con Kotlin cuando se desee generar Javascript. Con código Kotlin puro se puede generar Javascript directamente sin necesidad de GWT. Es probable que con IKVM (**ver Nota 2**) se pueda generar bitcode de la plataforma .NET, incluyendo Windows 10 y Windows Phone y Unity.

Nota 1: GWT o Google Web Toolkit es un framework creado por Google que permite ocultar la complejidad de varios aspectos de la tecnología AJAX.
GWT es un:

- Generador de Javascript: convierte código java a javascript.
- Compilador, preprocesador, linker y optimizador completo (no sólo compresión y ofuscación).
- SDK (kit de desarrollo).
- Un conjunto de herramientas para testear, depurar, estadísticas, etc. Dado que proporciona un conjunto de módulos: DOM, XML, I18N, JSON, RPC, y Widgets básicos, y mucho mas: css sprite, image bundling y otros.
- Es código Java para ejecutarse en la máquina virtual 'browser' en lugar de sobre el 'jre'.
- El código generado que puede incluirse en cualquier HTML (estático o dinámico), y puede interaccionar con éste (modificar el DOM, ejecutar AJAX, validar formularios, por lo que es compatible con SEO y 'progressive enhancement'.

Nota 2: IKVM.NET es una implementación de Java para Mono y también para Microsoft .NET Framework. IKVM es software libre, distribuido bajo *permissive free software licence* e incluye los siguientes componentes:

- Una Máquina Virtual de Java implementada en .NET.
- Una implementación de biblioteca de clases Java en .NET.
- Herramientas de interoperabilidad entre Java y .NET.
- Ejecutar directamente código compilado en Java y directamente en Microsoft .NET o Mono incluyendo también MonoDevelop. El código es convertido a CIL (Common Intermediate Language) y es ejecutado.

Algunas API de Android, como Android KTX (ver **Nota 3**), son específicas de Kotlin, pero la mayoría están escritas en Java y se pueden llamar desde Java o Kotlin. La interoperabilidad de Kotlin con Java implica que no se tiene que implantar Kotlin de una sola vez; es decir, se pueden tener proyectos con código Kotlin y Java.

Nota 3: KTX es un conjunto de extensiones de Kotlin para el desarrollo de aplicaciones de Android.

- El objetivo de Android KTX es aprovechar las características del lenguaje. Por ejemplo, funciones/propiedades de extensión, lambdas, parámetros con nombre y valores predeterminados de parámetros. El objetivo claro de este proyecto es para no agregar nuevas características a la API de Android existente.

Sintaxis de Kotlin

Algunas de las características básicas de la codificación en Kotlin son las siguientes:

- Los `;` (punto y coma, o semicolon) son opcionales.

- Posee estructuras del lenguaje para trabajar con tipos nulables.

```
var a:String = "" a = null; // No se permite var b:String? = "" b = null; // Permitido
```

- Con respecto a Java, en Kotlin el operador ternario no existe.

```
fun clamp(v:Int, min:Int, max:Int) = if (v < min) min else if (v > max) max else v
```

- Tiene smart casts basados en las ramas de código como TypeScript, en `if` y en `when` (el equivalente a `match` y `switch` de otros lenguajes):

```
fun mymethod(a:Any) {
    if (a is String) {
        println(a.toUpperCase()) // Se puede poner toUpperCase porque a es de tipo cadena
    }
}
```

- Interpolación avanzada de cadenas de expresiones completas:

```
fun hello(name:String) = "Hello $name"
fun returnDoublesAsString(items:Iterable<Int>) = "Sum is ${ items.map { it * 2 }.join(", ")
}"
```

- Admite funciones globales a nivel de paquete (que, en el caso de la JVM, acaban como parte de una clase llamada como el nombre del archivo con el sufijo `Kt`).

- Posee tipos `object` que acaban siendo singleton y que admiten herencia en vez de métodos estáticos. Utiliza `companion object` para hacer clases con mezcla de elementos estáticos y de instancia. Soporta la anotación `@JvmStatic` para generar métodos estáticos para poder interoperar mejor con Java.

```
class Test {
    companion object {
        fun staticMethod() { }
    }
    val instanceConst = 10
}
```

- Permite crear value objects en una sola línea y la omisión de las llaves { } cuando no hace falta:

```
data class MyClass(val a:Int, val b:Int, val c:Int)
```

- Admite funciones de una línea sin return utilizando = en vez de llaves `{ }`, permitiendo en este caso omitir el tipo de retorno.

```
fun sum(a:Int, b:Int) = a + b
```

- No hay distinción entre tipos primitivos y clases. La primera consecuencia directa de esto es que incluso los tipos primitivos empiezan por mayúscula por convención: `Byte`, `Char`, `Int`, `Long`, `Float`, `Double`, `Boolean`.

- Utiliza las declaraciones de tres letras: `var`, `val` y `fun`. Usa `var` para declaraciones mutables, `val` para declaraciones inmutables y `fun` para funciones.

```
var mutable = 1 val immutable = 2 fun myfunc() { }
```

- Admite tipos y funciones anónimas e internas. Admite varias declaraciones de tipo por archivo. Se pueden incluir varias clases y funciones libres de paquete en un solo archivo. Los argumentos son inmutables, evitando que se pueda incurrir en cierto tipo de errores:

```
fun plusOne(a:Int) {
    a++ // ¡No!
    return a
}
```

- Posee `getters` y `setters` reales que permiten en mayor medida la supresión de tipos, es decir, eliminación de la vocal con que acaba una palabra cuando la siguiente empieza por vocal:

```
private var myValue:Int = 1
val immutableDouble:Int get() = myValue * 2
var double:Int
    get() = myValue * 2
    set(value) { myValue = value / 2 }
```

- Tiene métodos de extensión libres y se importan individualmente (aunque también se pueden importar junto al paquete con imports *), el IDE se encarga del importado, reduciendo el coste en el lado del compilador.

- Si se utilizan secuencias.

```
fun Int.double() = this * 2 val v1 = 10 val v2 = v1.double()
```

Desarrollo

1. Crear un nuevo proyecto sin actividad; es decir, seleccionar No Activity, como se muestra en la figura siguiente.

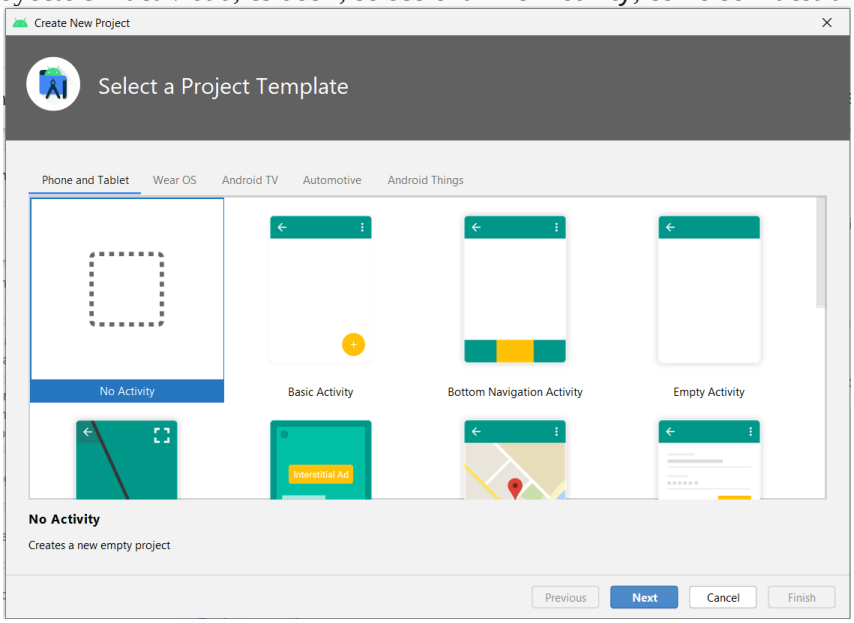


Figura 1. La selección No Activity.

2. La opción Minimum SDK puede tener cualquier valor y no afectará el resultado.

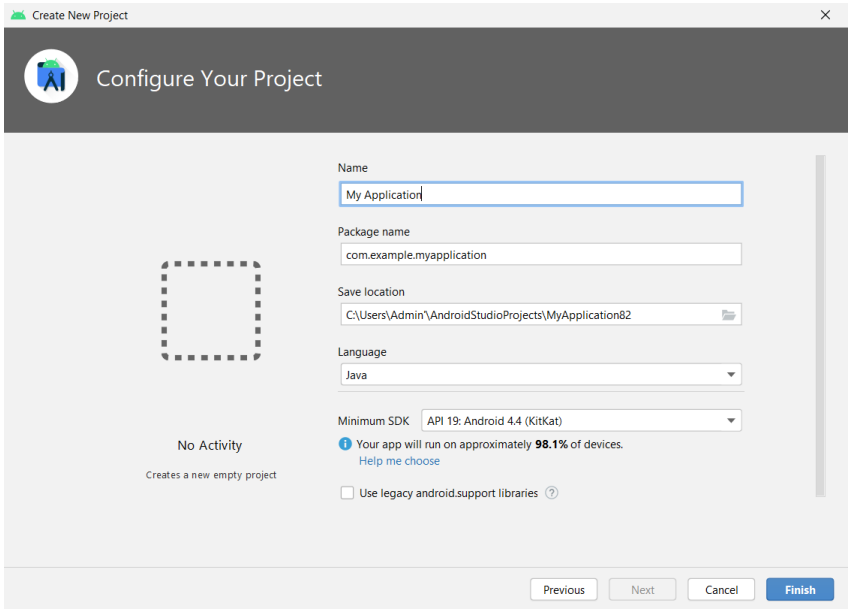


Figura 2. El Minimum SDK seleccionado no afecta el resultado.

3. Crear un objeto de modelo `User`, y una clase singleton `Repository` que trabaje con objetos `User` y muestre listas de usuarios y nombres de usuarios formateados.

   Crear un nuevo archivo denominado `User.java` en app/java/<mipaquete> y pegar el siguiente código:

```
public class User {
    @Nullable
    private String firstName;
    @Nullable
    private String lastName;
    public User(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}
```

Observar que el IDE indica que `@Nullable` no está definido androidx.annotation.Nullable? Alt+Intro. Por lo tanto, se debe importar `androidx.annotation.Nullable`.

4. Crear un nuevo archivo denominado `Repository.java` y pegar el siguiente código:

```
import java.util.ArrayList;
import java.util.List;
```

```
public class Repository {
    private static Repository INSTANCE = null;
    private List<User> users = null;
    public static Repository getInstance() {
        if (INSTANCE == null) {
            synchronized (Repository.class) {
                if (INSTANCE == null) {
                    INSTANCE = new Repository();
                }
            }
        }
        return INSTANCE;
    }
    // keeping the constructor private to enforce the usage of getInstance
    private Repository() {
        User user1 = new User("Jane", "");
        User user2 = new User("John", null);
        User user3 = new User("Anne", "Doe");
        users = new ArrayList();
        users.add(user1);
        users.add(user2);
        users.add(user3);
    }
    public List<User> getUsers() {
        return users;
    }
    public List<String> getFormattedUserNames() {
        List<String> userNames = new ArrayList<>(users.size());
        for (User user : users) {
            String name;
            if (user.getLastName() != null) {
                if (user.getFirstName() != null) {
                    name = user.getFirstName() + " " + user.getLastName();
                } else {
                    name = user.getLastName();
                }
            } else if (user.getFirstName() != null) {
                name = user.getFirstName();
            } else {
                name = "Unknown";
            }
            userNames.add(name);
        }
        return userNames;
    }
}
```

5. Declaración de `nullabiltiy`, `val`, `var` y clases de datos.

El IDE puede convertir automáticamente el código Java en código Kotlin. El IDE realiza un pase inicial de la conversión, pero luego se analizará el código resultante para comprender cómo y por qué se ha convertido de esta manera. En futuras versiones de Android Studio el convertidor automático puede generar resultados diferentes.

Seleccionar el archivo User.java para convertirlo a Kotlin. En la barra de menú:
Barra de menú>Code>Convert Java File to Kotlin File 

Si Kotlin no se encuentra configurado, se solicitará la configuración como se indica en las siguientes figuras:

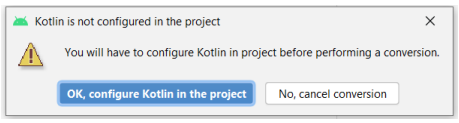


Figura 3. Kotlin no esta configurado.

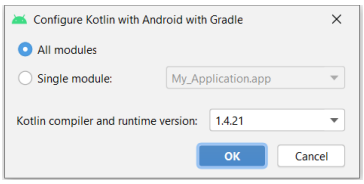


Figura 4. Digitar OK.

Durante la conversión en el editor se pueden mostrar los siguientes mensajes:

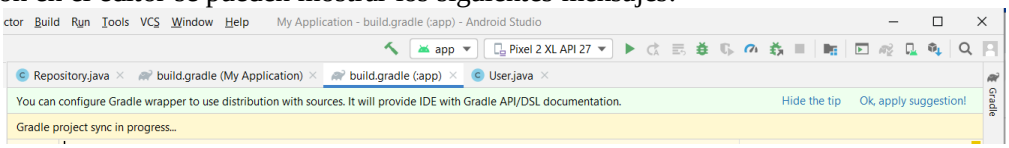


Figura 5. La sincronía del proyecto Gradle está en progreso.

Si no se encuentra actualizada la conversión a Kotlin, se mostraría el siguiente mensaje. Seleccionar la opción Update:

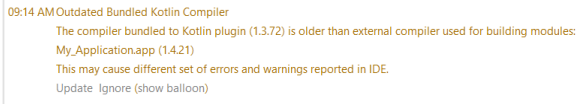


Figura 6. El compilador de Kotlin se encuentra obsoleto.

Enseguida se procede a la descarga e instalación del software de actualización de Kotlin por medio del IDE. Al terminar la instalación digitar OK:

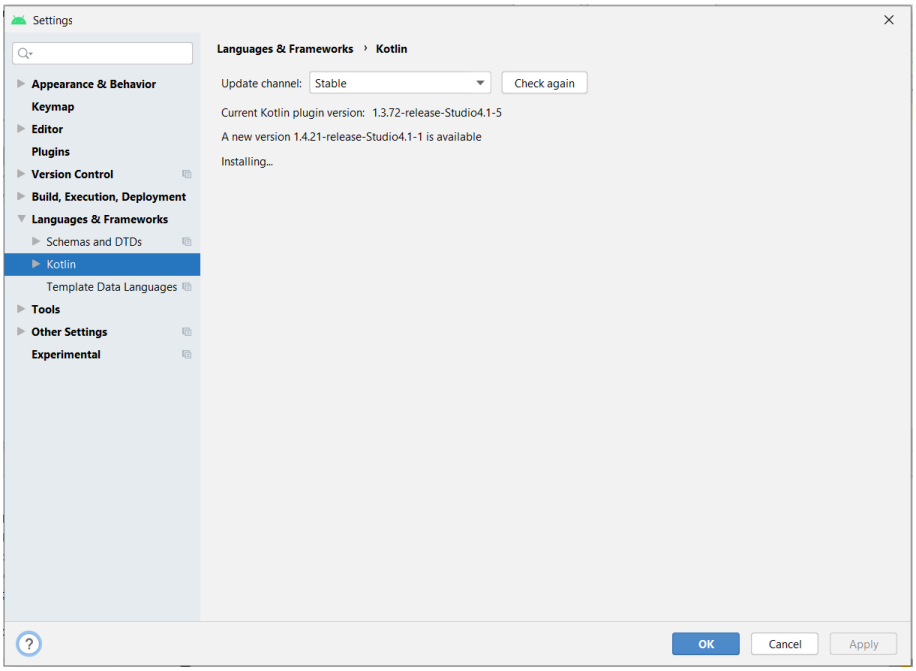


Figura 7. La ventana Settings de Kotlin.

Si el IDE solicita alguna corrección después de la conversión, seleccionar Yes.

Debido a los cambios realizados, se debería ver el siguiente código de Kotlin:

```
class Usuario (var firstName: String ?, var lastName: String?)
```

Observar que el archivo User.java fue renombrado como User.kt. Los archivos Kotlin tienen la extensión .kt.

NOTA: Si pega código Java en un archivo Kotlin, el IDE convertirá automáticamente el código pegado a Kotlin.

En la clase `User` de Java se tenían dos propiedades: `firstName` y `lastName`. Cada uno tenía un método `getter` y `setter`, lo que hacía que su valor fuera mutable. La palabra clave de Kotlin para las variables mutables es `var`, por lo que el convertidor usa `var` para cada una de estas propiedades. Si las propiedades de Java solo tuvieran `getters`, serían inmutables y se habrían declarado como variables `val`. La palabra `val` es similar a la palabra clave `final` en Java.

Una de las diferencias clave entre Kotlin y Java es que Kotlin especifica explícitamente si una variable puede aceptar un valor `null`. Lo hace agregando un `` `?` `` a la declaración de tipo.

Debido a que se marcan `firstName` y `lastName` como anulables, el convertidor automático marcó automáticamente las propiedades como anulables con `String ?`. Si se anotan sus miembros de Java como no nulos (usando `androidx.annotation.NonNull`), el convertidor reconocerá esto y hará que los campos no sean nulos en Kotlin también.

NOTA: En Kotlin, se recomienda usar objetos inmutables siempre que sea posible (es decir, usar `val` en lugar de `var`) y evitar los tipos que aceptan valores `NULL`. Se debe hacer que la nulabilidad sea significativa de algo que se desee manejar específicamente. Hasta este momento, la conversión básica ya está realizada.

La Clase de Datos

La clase de `User` sólo contiene datos. Kotlin tiene una palabra clave para las clases con este rol: `data`. Al marcar esta clase como una clase de datos, el compilador creará automáticamente métodos `getters` y `setters`. También los métodos `equals` (), `hashCode` () y `toString` ().

Se agrega la palabra clave `data` a la clase `User`:

```
data class User (var firstName: String, var lastName: String)
```

Kotlin, al igual que Java, puede tener un constructor principal y uno o más constructores secundarios. El constructor del ejemplo anterior es el constructor principal de la clase `User`. Si se está convirtiendo una clase Java que tiene varios constructores, el convertidor también creará automáticamente varios constructores en Kotlin. Se definen mediante la palabra clave `constructor`.

Si se desea crear una instancia de esta clase, se puede hacer así:

```
val user1 = Usuario ("Jane", "Doe")
```

Equality.

Kotlin tiene dos tipos de igualdad:

- La igualdad estructural utiliza el operador == y llama a `equals` () para determinar si dos instancias son iguales.
- La igualdad referencial utiliza el operador === y comprueba si dos referencias apuntan al mismo objeto.

Las propiedades definidas en el constructor principal de la clase de datos se usarán para verificaciones de igualdad estructural.

```
val user1 = User("Jane", "Doe")
val user2 = User("Jane", "Doe")
val structurallyEqual = user1 == user2 // true
val referentiallyEqual = user1 === user2 // false
```

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.

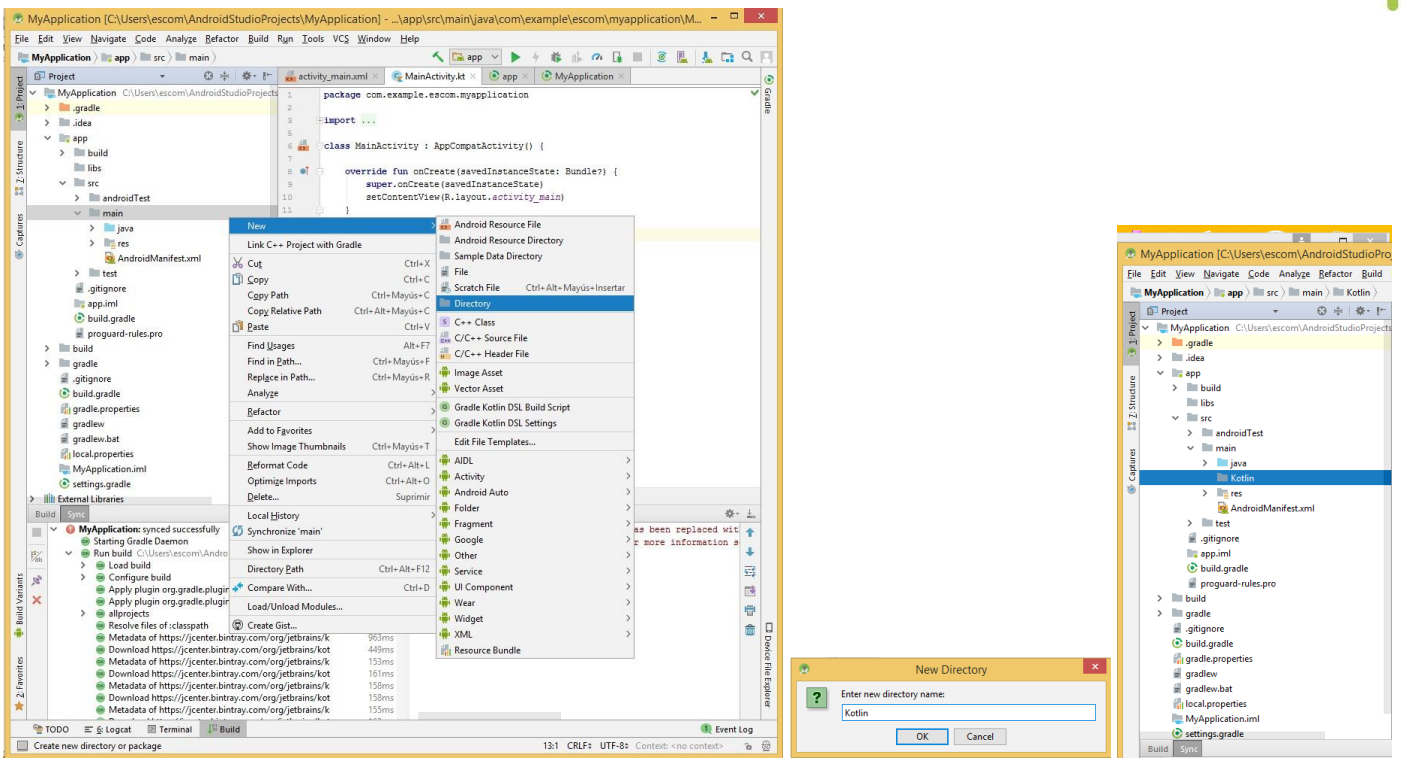
El resto de esta `Activity` es bastante similar a Java. Sin embargo, las siguientes líneas muestran otra característica clave de Kotlin:

```
super.onCreate(savedInstanceState)
setContentView(R.layout.activity_main)
```

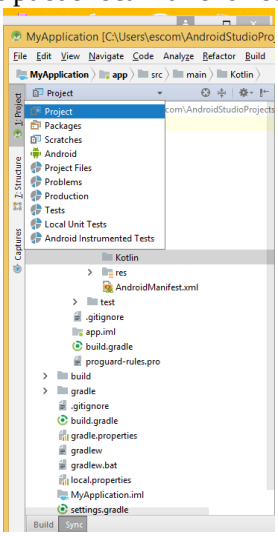
En Kotlin no se necesita que las líneas terminen con punto y coma, de ahí la ausencia de dos puntos en el fragmento anterior. Si se desea se pueden agregar dos puntos, pero el código será más limpio y fácil de leer sin ellos.

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.



Si no se detecta el directorio principal del proyecto, abrir el pequeño menú desplegable en la esquina superior izquierda del Project Explorer y seleccionar Project. Ahora se puede localizar el directorio `src/main`.



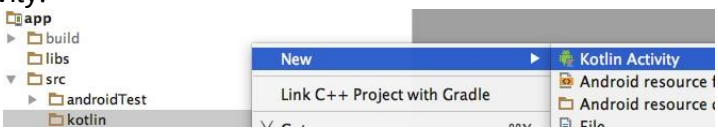
Una vez creado la carpeta dedicada a Kotlin, arrastrar el archivo MainActivity.kt dentro de él. Asegurarse de conservar el nombre del paquete existente del archivo MainActivity.kt para que el proyecto aún se ejecute.

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.

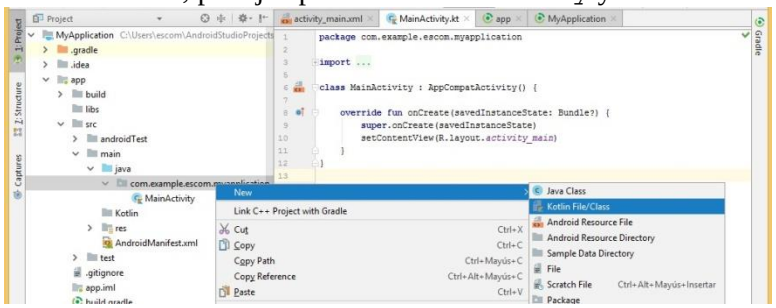
Dado que Kotlin compila en *bytecode*, una aplicación escrita en Kotlin se siente exactamente igual que una aplicación escrita en Java, así que si se instala esta aplicación en un dispositivo Android o en un AVD compatible, debería parecer como si nada hubiera cambiado.

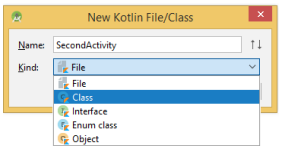
Creación de archivos adicionales de Kotlin.
Si en el proyecto se continúa trabajando con Kotlin, seguramente se necesitará crear nuevos archivos de Kotlin en lugar de simplemente convertir los existentes de Java.

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:

```
package com.example.escom.myapplication

class SecondActivity {
}
```

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:

```
import android.app.Activity
import android.os.Bundle
import android.app.Activity
```

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

```
class SecondActivity: Activity () {

}
```

Después, sobrecargar el método `onCreate` de la actividad:

```
override fun onCreate (savedInstanceState: Bundle?) {
      super.onCreate (savedInstanceState)
}
```

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:

```
<activity
```

```
      android:name=".SecondActivity"
      android:label="@string/second"
      android:parentActivityName=".MainActivity">
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity" />
  </activity>
```

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'
```

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 kotlinx.android.synthetic.main.<layout>.<view-id>
```

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 kotlinx.android.synthetic.main.activity_main.textView1
```

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 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="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context="com.jessicathornsby.myapplication.MainActivity">

  <TextView
      android:id="@+id/myTextView"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
       />

</RelativeLayout>
```

Luego se importa el `TextView` en el MainActivity.kt, y se asigna su texto utilizando solamente su ID:

```
import android.support.v7.app.AppCompatActivity
import android.os.Bundle
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")
  }
}
```

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

```
import kotlinx.android.synthetic.main.<layout>.*
```

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

```
kotlinx.android.synthetic.main.activity_main.*.
```

NOTA. Reportar la ejecución con imágenes de cada uno de los métodos y atributos indicados, en un documento AlumnoTarea32Grupo.pdf. Enviarlo al sitio indicado por el profesor.