INTRODUCCIÓN

Para desarrollar aplicaciones con UI (User Interface) o Interfaz de Usuario, ha creado Android un kit de herramientas para simplificar el desarrollo de las interfaces y aprovechar la nueva tecnología de teléfonos inteligentes o smartphones plegables. Ya que, con el anterior sistema de desarrollo en Android Studio se hubiera complicado mayormente el desarrollo de las aplicaciones.

Además, para aderezar la transición escogieron un lenguaje de código abierto, de programación funcional y orientado a objetos con calidad industrial. Es el lenguaje Kotlin, que al poseer una sintaxis más sencilla que Java y recordando a Scala permite una interoperabilidad con Java y que los desarrolladores puedan ser más productivos sin renunciar a sus programas creados en Java o aprovechar funcionalidades de Java que pueden ser invocadas en el mismo programa de Kotlin sin interferir entre los dos lenguajes.

En consecuencia, para introducir nuevos programadores se ha creado esta obra, que permitirá a los programadores noveles desarrollar aplicaciones para el sistema operativo Android, que actualmente es el de mayor uso en el mundo en los smartphones.

Para conseguir que el lector adquiera los conocimientos de programación, el libro tiene las siguientes metas:

a) Mostrar de forma sencilla los fundamentos de la programación: la secuencia, la bifurcación y la repetición.

b) Enseñar como se interactúa con la interfaz gráfica de Android.

c) Comprender el uso de estructuras de datos y la programación orientada a objetos.

En todo momento, el lector se vera apoyado por numerosos ejemplos prácticos, los cuales son realizados paso a paso para que el lector los comprenda completamente.

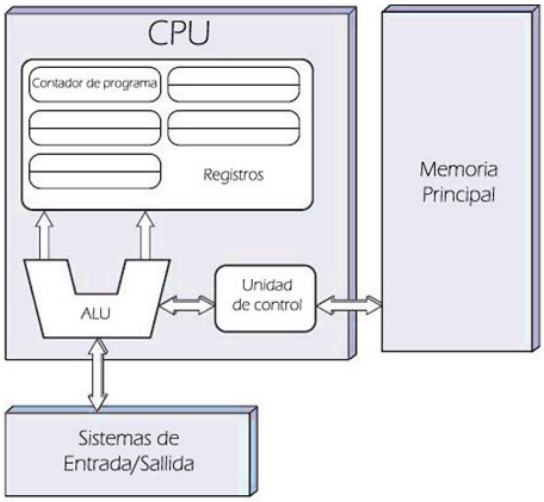
1

INTRODUCCIÓN A LA PROGRAMACIÓN

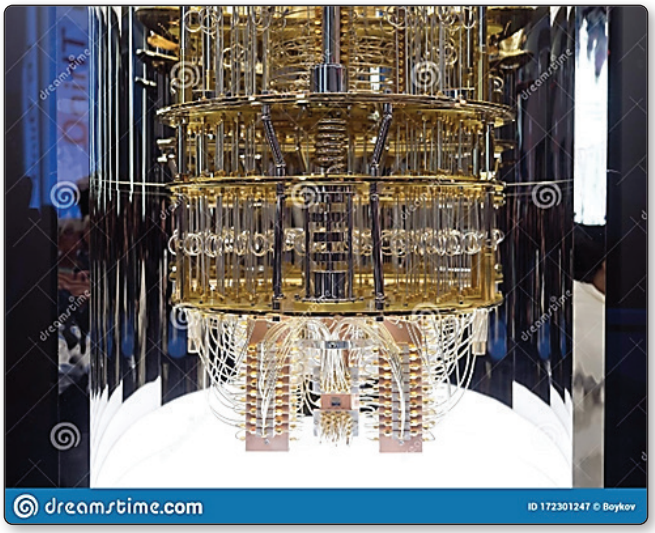
1.1 INTRODUCCIÓN

El diccionario de la RAE define programar como "Idear y ordenar las acciones necesarias para realizar un proyecto", y es la definición que más se aproxima a la hora de realizar el acto de construir unas instrucciones para que las ejecute una CPU.

Desde los primeros ordenadores como el Colossus o ENIAC hasta los ordenadores miniaturizados en Smartphones hasta los ordenadores cuánticos, existe una necesidad de organizar y estructurar los elementos existentes en estos ordenadores para conseguir un fin. Este puede ser desde realizar una tarea tan sencilla como sumar, escribir un mensaje o clasificar una imagen y tomar ciertas decisiones, o simplemente ejecutar un programa de desencriptación que en un ordenador clásico tardaría mucho tiempo, en otro, utilizando diferentes algoritmos cuánticos tardará muchísimo menos.

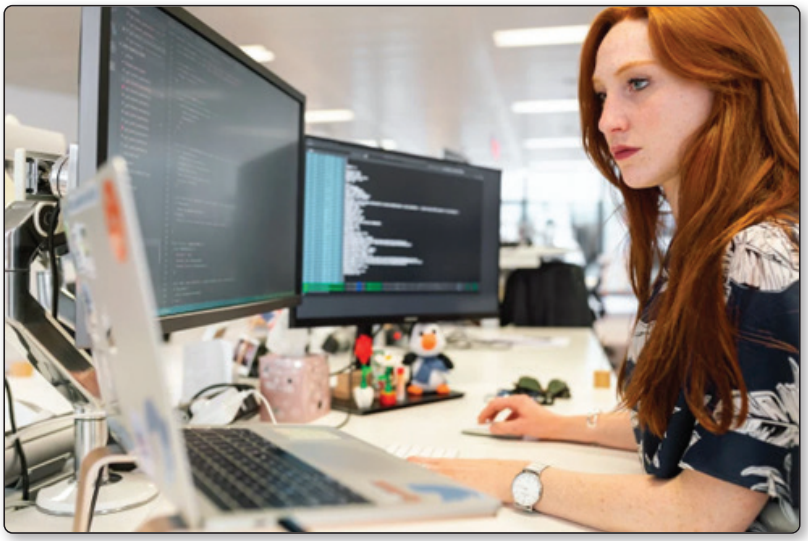


[2] Fuente imagen de Wikipedia



[3] Fuente dreamstime.com

Sin embargo, en esta pequeña introducción de la historia de la informática tenemos dos elementos que se deben conjugar: **el hardware y el software**. Para ello, se creó la disciplina de **ingeniería del software** es una de las ramas de las ciencias de la computación que estudia la creación de software confiable y de calidad, basándose en métodos y técnicas de ingeniería, y brindando soporte operacional y de mantenimiento. El campo de estudio de la ingeniería de software integra **ciencias de la computación**, **ciencias aplicadas** y **las ciencias básicas** en las cuales se encuentra apoyada la ingeniería.[1]



[4] Fuente pexels.com

Las definiciones más reconocidas, formuladas por los siguientes prestigiosos autores:

- Ingeniería de software es el estudio de los principios y metodologías para el desarrollo y mantenimiento de sistemas software (Zelkovitz, 1978).
- Ingeniería de software es la aplicación práctica del conocimiento científico al diseño y construcción de programas de computadora y a la documentación asociada requerida para desarrollar, operar y mantenerlos. Se conoce también como desarrollo de software o producción de software (Bohem, 1976).
- La ingeniería de software trata del establecimiento de los principios y métodos de la ingeniería a fin de obtener software de modo rentable, que sea fiable y trabaje en máquinas reales (Bauer, 1972).
- La ingeniería de software es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación, y mantenimiento del software. *Standard Glossary of Software Engineering Terminology.*

En la Ingeniería del software una de las decisiones de vital importancia es la elección del lenguaje de programación.

La definición de lenguaje de programación es un conjunto de reglas tanto sintácticas como semánticas que le dicen al ordenador en qué orden deben ejecutarse instrucciones para alcanzar un objetivo.

Por lo tanto, para la elección del lenguaje de programación se deberán establecer los requisitos por parte de la dirección del proyecto para seleccionar el lenguaje idóneo, en la elaboración del proyecto y su consecución. Cada lenguaje está diseñado con propósitos específicos y funcionalidades preparadas para ello.

Dicha elección deberá tenerse en cuenta estos parámetros:

- *Compilados e interpretados*: los lenguajes compilados son más rápidos que los interpretados ya que estos deben ejecutarse en un entorno virtual, aparte de leerse línea a línea se ejecutan en un entorno preparado para ello. Como consecuencia, son más lentos que los lenguajes compilados que se leen una vez, se genera el archivo ejecutable, y al estar preparados para ejecutarse en lenguaje máquina son más rápidos.
- *Multiplataforma*: estos lenguajes no les importa en qué sistema operativo se ejecuten, como tienen un entorno virtual de ejecución, cuando se realiza la interpretación se genera un lenguaje intermedio que es el bytecode. Este puede ejecutarse en varios sistemas operativos ya que dicho entorno virtual les abstrae de los sistemas operativos que se encuentran ejecutándose en sus máquinas virtuales.
- *Seguridad*: hay lenguajes que debido a su evolución encadenan, y además, no aprovechan los últimos avances de seguridad que ofrecen las CPU. El caso más destacado es un nuevo lenguaje de programación llamado Rust que aprovecha los últimos avances de seguridad de la CPU y está intrincado en el propio ADN de dicho lenguaje.

Se pueden hablar de muchas características de diferentes lenguajes de programación, con sus pros y sus contras. Dependiendo del objetivo a conseguir se debería realizar una tabla con las columnas correspondientes y seleccionar qué lenguaje se adapta mejor a los propósitos para el desarrollo de la aplicación que se desea construir.

Solamente se ha mencionado las tres características fundamentales, pero se deberían tener en cuenta más características como:

- Curva de aprendizaje.
- Soporte a largo plazo.
- Experiencia de los programadores.
- Librerías que tienen soporte y mantenimiento.
- Facilidad para crear librerías.
- Nivel de abstracción en el lenguaje:
  - Lenguaje primera generación: programación en binario, es trabajar directamente con el lenguaje que habla el propio ordenador.
  - Lenguaje segunda generación: lenguaje ensamblador, agrupa secuencias de 0 y 1 para elaborar sentencias.
  - Lenguaje tercera generación: lenguajes parecidos al del ser humano, pero con muchas restricciones. Se pueden citar: Python, C/C++, C#, Java, Kotlin, Rust. Estos son lenguajes imperativos, indican los pasos que se deben seguir para obtener los resultados.
  - Lenguaje cuarta generación: son lenguajes declarativos dicen lo que quieren obtener, pero no cómo hacerlo. El ejemplo significativo es SQL, y se añadido también Jetpack Compose en la declaración de las pantallas en el entorno de Android.
  - Lenguaje quinta generación: lenguajes que son capaces de entender al desarrollador y analista de aplicaciones para crear las aplicaciones que le indican con condiciones y necesidades. Se necesita inteligencia artificial para llevarlo a cabo.

1.2 KOTLIN Y JETPACK COMPOSE

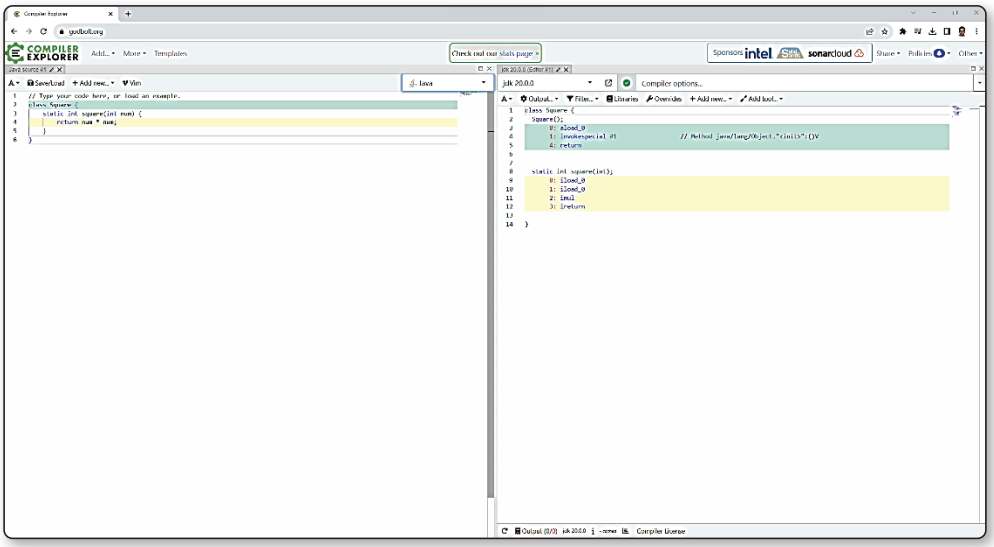
La elección del conjunto del lenguaje Kotlin y Jetpack Compose en IDE de Android Studio para el desarrollo de aplicaciones móviles y de multiplataforma.

La primera razón es que Kotlin y Jetpack Compose permiten ser más productivos a los desarrolladores de aplicaciones móviles. Para ello, escribir menos código afecta a todas las etapas de desarrollo: como autor, puedes enfocarte en el problema en cuestión, dado que debes realizar menos pruebas y depuraciones, y se reduce el margen de error; como revisor o compilador, tienes menos código para leer, comprender, revisar y mantener [5].

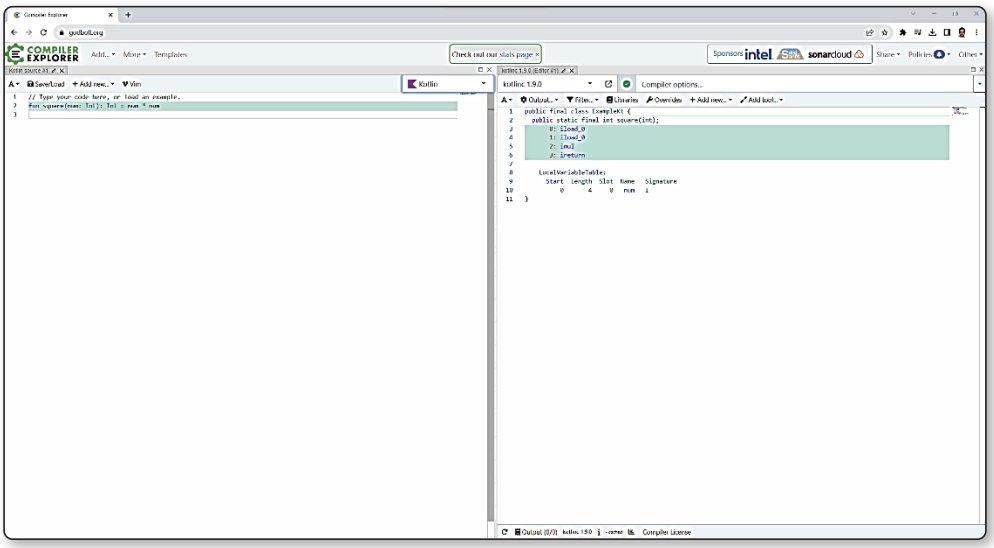
La cantidad de código que genera Kotlin es un 30% menos que Java, si además, se combina con Compose se entiende que el código solamente se escribe en Kotlin, en lugar de dividirlo entre Kotlin y XML. Es mucho más fácil revisar el código cuando está escrito en el mismo lenguaje y, a veces, en el mismo archivo, sin necesidad de alternar entre Kotlin y XML [5].

Las herramientas modernas de Android para compilar IU nativas. Simplifica y acelera el desarrollo de IU en Android para que tus apps cobren vida con herramientas potentes, API intuitivas de Kotlin y menos código. Compila IU de Android de manera más fácil y ágil [5].

La segunda razón es que Kotlin genera un bytecode más limpio que Java para la misma JVM (Java Virtual Machine):



Bytecode generado para Java. Fuente https://godbolt.org/



Bytecode generado para Kotlin. Fuente https://godbolt.org/

Se puede apreciar que es mucho mejor la generación de Bytecode para Kotlin.

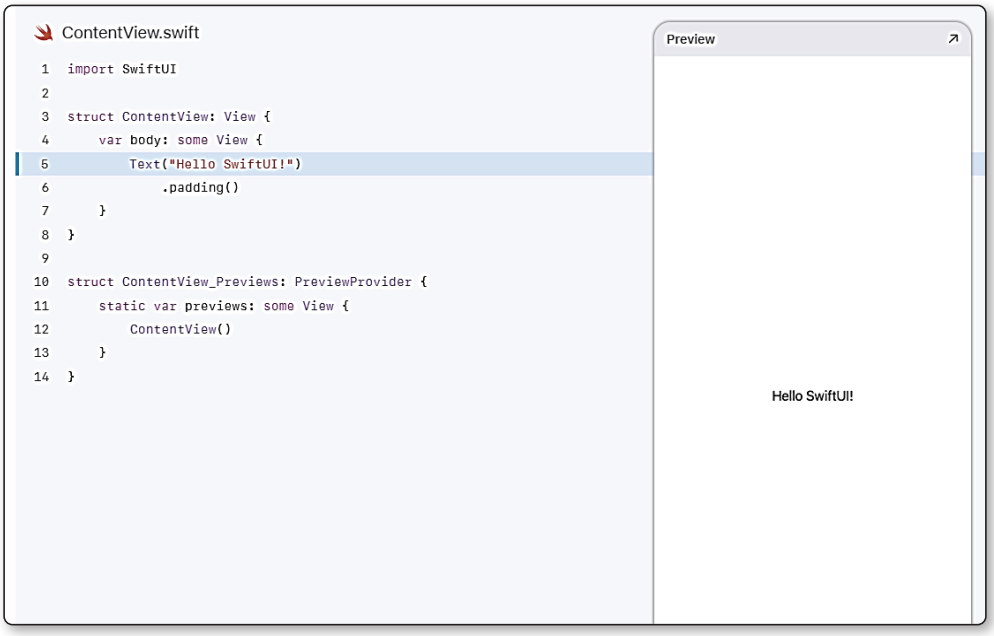
La tercera razón: Android quiere ir migrando de Java a Kotlin en un futuro. Por lo tanto, empezar con un lenguaje de futuro para el desarrollo de aplicaciones móviles es lo mejor para que comiencen los programadores noveles y es muy recomendable que los programadores con experiencia vayan migrando de Java a Kotlin.

La cuarta razón es a nivel pedagógico, los lenguajes de programación tienden a la evolución convergente todos se van pareciendo e igual que la evolución biológica convergen en soluciones parecidas debido a la experiencia de los desarrolladores del lenguaje y la retroalimentación de los creadores de aplicaciones.

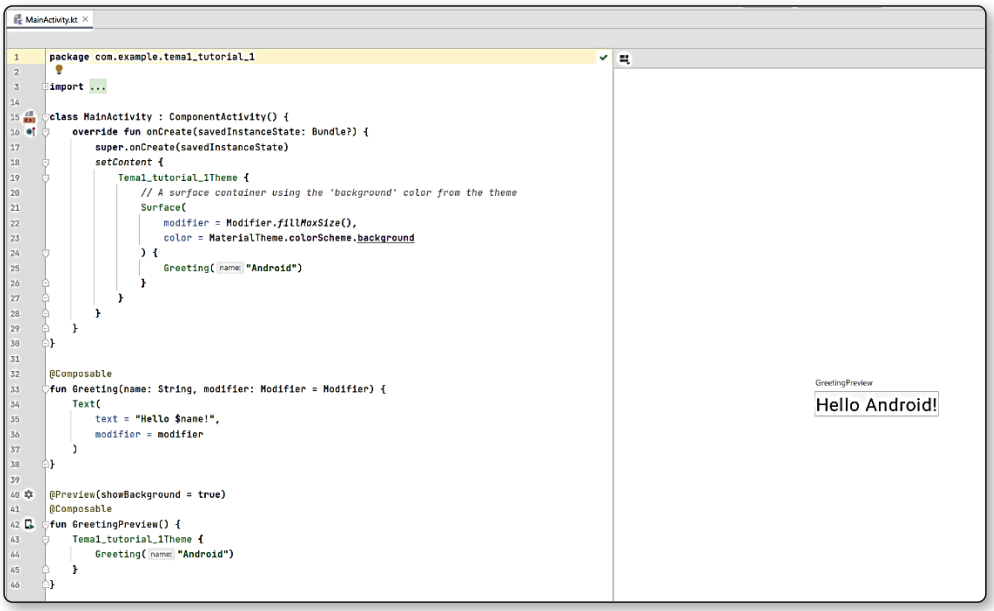
lib/main.dart

```dart
import 'package:english_words/english_words.dart';
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return ChangeNotifierProvider(
      create: (context) => MyAppState(),
      child: MaterialApp(
        title: 'Namer App',
        theme: ThemeData(
          useMaterial3: true,
          colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepOrange),
        ),
        home: MyHomePage(),
      ),
    );
  }
}

class MyAppState extends ChangeNotifier {
  var current = WordPair.random();
}

class MyHomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    var appState = context.watch<MyAppState>();

    return Scaffold(
      body: Column(
        children: [
          Text('A random idea:'),
          Text(appState.current.asLowerCase),
        ],
      ),
    );
  }
}
```

Mostrar en Flutter. Fuente https://docs.flutter.dev/get-started/codelab



Mostrar texto en Swift. Fuente https://developer.apple.com/tutorials/swiftui/creating-and-combining-views



Mostrar texto en Jetpack Compose. Fuente https://developer.android.com/jetpack/compose/tutorial?hl=es-419

Como se aprecia en la comparativa de las tres muestras de texto en Compose, Flutter y Swift se parecen mucho las técnicas empleadas. Como consecuencia aprender Kotlin y JetPack Compose facilita la migración entre estas tres plataformas de desarrollo y la interoperabilidad.

1.3 BIBLIOGRAFÍA

[1] *https://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_software*

[2] *https://es.wikipedia.org/wiki/Arquitectura_de_Von_Neumann*

[3] *https://es.dreamstime.com/ibm-q-system-one-quantum-computer-en-consumer-electronic-show-ces-las-vegas-nevada-de-enero-image172301247#_*

[4] *https://www.pexels.com/es-es/foto/mujer-codificacion-en-computadora-3861958/*

[5] *https://developer.android.com/jetpack/compose/why-adopt#intuitive*