Introduccion a Kotlin.pdf

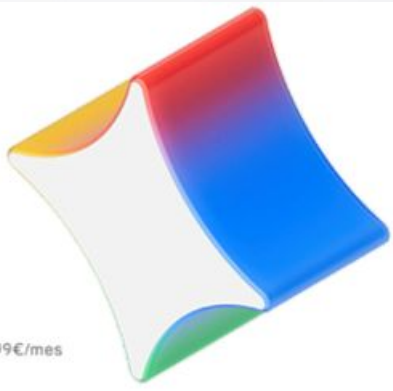
quiico_

Programación multimedia y dispositivos móviles

1º Desarrollo de Aplicaciones Multiplataforma

Estudios España

Accede al documento original

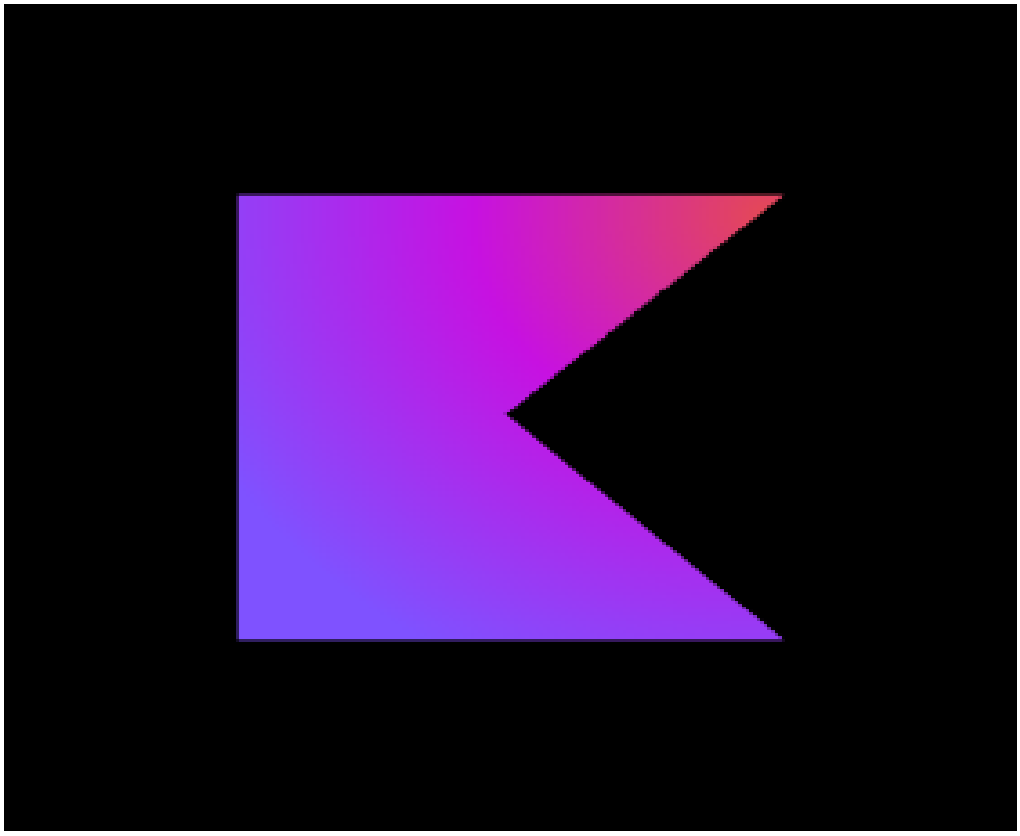


Introducción a Kotlin

2º DAM Curso 2024/2025

Asignatura: Programación Multimedia y Dispositivos Móviles

- **Profesor:** Pablo Rodríguez
- **Alumno:** Francisco José Alonso de Caso Ortiz Nº2



ÍNDICE

Introducción

¿Qué es Kotlin?

Es un lenguaje de programación estático de código abierto que admite la programación funcional y orientada a objetos. Corre sobre la máquina virtual de Java.

Que un lenguaje de programación sea estático quiere decir que el **sistema de tipos** del lenguaje está estáticamente verificado. Esto significa que el tipo de las variables, funciones o expresiones se determina **en tiempo de compilación**, antes de ejecutar el programa.

Estático → Se comprueba siempre antes de ejecutarse en el proceso de compilación

En otras palabras:

1. **Verificación anticipada**: Los tipos se comprueban cuando el código se compila, lo que permite detectar errores de tipo (cómo asignar un número a una variable que debería contener un texto) antes de ejecutar el programa.
2. **Tipos fijos**: Una vez que se declara el tipo de una variable, no se puede cambiar a otro tipo. Por ejemplo, si declaras que una variable es de tipo entero (número), no podrás asignarle un valor de tipo texto más adelante.
3. **Seguridad de tipos**: Al forzar la declaración de tipos desde el principio, se evitan errores comunes en tiempo de ejecución, haciendo que el programa sea más robusto.

En contraste, en los lenguajes **dinámicos** (como Python o JavaScript), los tipos se verifican en tiempo de ejecución, lo que permite más flexibilidad, pero también puede llevar a errores no detectados hasta que el código ya esté en ejecución.

¿Por qué Kotlin en lugar de Java?

Interoperable con Java lo que significa que puede trabajar perfectamente con el código y las bibliotecas escritas en Java. Esta interoperabilidad es clave en el desarrollo de aplicaciones móviles, especialmente para Android, ya que:

- **Reutilización de código**: Si ya tienes proyectos o bibliotecas en Java, no necesitas reescribir todo desde cero. Kotlin puede integrarse sin problemas, facilitando la transición.
- **Ecosistema Android**: Como Android fue construido inicialmente sobre Java, Kotlin se beneficia del extenso ecosistema de herramientas y bibliotecas de Java sin perder funcionalidad.

Simpleza y estabilidad → Kotlin está diseñado para reducir la verbosidad de Java, lo que hace que el código sea más limpio y fácil de leer.

- **Menos código boilerplate**: En Kotlin, muchas tareas comunes como declarar variables, getters/setters, o manejar excepciones se pueden hacer con mucho menos código. Esto simplifica el desarrollo multimedia y de aplicaciones móviles, que suelen requerir cambios rápidos y mantenibilidad.

- **Estabilidad**: Al ser más conciso y explícito que Java, se reducen los errores humanos, lo que resulta en aplicaciones más estables.

Seguro

- **Null Safety**: Kotlin tiene un sistema que reduce la posibilidad de que ocurran errores de referencia nula, lo cual es vital en el desarrollo móvil donde las aplicaciones deben ser robustas y confiables.
- **Evitar fallos**: Esta característica ayuda a evitar fallos inesperados que pueden resultar en aplicaciones que se cuelgan o dejan de funcionar, lo que es crítico en aplicaciones móviles donde la experiencia del usuario es fundamental.

No penaliza el rendimiento

- **Compilación a bytecode de JVM**: Al igual que Java, Kotlin se compila a **bytecode de JVM**, por lo que su rendimiento es comparable al de Java. Esto es crucial en aplicaciones multimedia, donde se requieren tiempos de respuesta rápidos y un procesamiento eficiente de recursos.
- **Optimización del código**: Kotlin está optimizado para que, a pesar de ser más conciso y menos propenso a errores, el rendimiento no se vea afectado, lo que es fundamental en el desarrollo de aplicaciones móviles que demandan eficiencia.

Multiplataforma (Kotlin-JS, Kotlin-Native, etc) →Kotlin no solo se limita a la JVM, sino que también permite la compilación a diferentes plataformas

- **Kotlin/JS**: Permite compilar código Kotlin a JavaScript, lo que facilita la creación de aplicaciones web en combinación con aplicaciones móviles.
- **Kotlin/Native**: Facilita la ejecución de código Kotlin en plataformas sin JVM, como iOS. Esto es especialmente importante para quienes quieren desarrollar aplicaciones móviles multiplataforma sin tener que escribir dos bases de código distintas (para Android e iOS).

  Esta característica permite que Kotlin sea una opción viable no solo para Android, sino también para el desarrollo multiplataforma, ahorrando tiempo y esfuerzo al unificar el código.

Conciso ya que está diseñado para ser más expresivo y menos prolijo que Java:

- **Menos código, más claridad**: La sintaxis concisa y directa de Kotlin permite escribir menos código para lograr lo mismo que en Java. Esto es particularmente valioso en la programación móvil, donde la velocidad de desarrollo y la claridad son importantes para cumplir con plazos y entregar productos de calidad.
- **Fácil mantenimiento**: Un código más corto y claro significa menos oportunidades de introducir errores y más facilidad para mantener y escalar las aplicaciones a medida que crecen.

Sencilla curva de aprendizaje

Kotlin es fácil de aprender para los desarrolladores que ya conocen Java, ya que está diseñado para ser compatible y familiar, pero a la vez ofrece mejoras significativas en términos de sintaxis y características avanzadas.

- **Rápida adopción**: Los desarrolladores de Java pueden migrar rápidamente a Kotlin sin enfrentarse a una curva de aprendizaje pronunciada, lo que permite comenzar a usar las ventajas de Kotlin de inmediato.
- **Amplia documentación**: Kotlin cuenta con una gran comunidad y excelente documentación, lo que facilita el aprendizaje y la resolución de problemas, algo esencial para equipos que desean migrar a un nuevo lenguaje en un entorno de desarrollo móvil.

Conceptos básicos

Declaración de clases e interfaces

Clase:

- Palabra reservada → class
- Para extender de otra clase o implementar una interfaz→ ":"
- Constructor por defecto vacío
- Código inicializador → init
- Para hacer extensible una clase → open / abstract

Interfaz:

- Palabra reservada → interface
- Contienen declaración de métodos, No implementaciones
- Una clase puede implementar una o más interfaces
- Pueden contener propiedades
- Pueden extender de otra interfaz → ":"

Variables y propiedades

En Kotlin encontramos los siguientes tipos básicos:

- Int
- Double
- Float
- Boolean
- String
- Declaradas como var (mutable) o val (inmutable)
- No es necesario especificar el tipo
- No es necesario crear getters o setters
- Constantes → const val
- Variables inicializadas de forma "tardía" → lateinit var
- Modificadores de visibilidad→ private, protected, internal y public

Funciones

- Palabra reservada → fun
- Parámetros de entrada → name: type

LA NUESTRA DURA MÁS

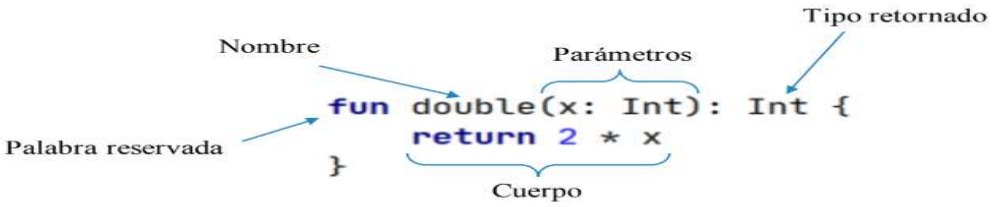
Nuestra tecnología y nuestra garantía, granuja...

5 AÑOS DE GARANTÍA

webuy.com

- Parámetros con valor por defecto
- Número variable de argumentos → vararg
- Si el método tiene una línea se pueden eliminar las llaves

Estructura



Nulabilidad

Si una variable puede ser nula y no es tratada debidamente el compilador lanzará un error
Declaración → var name: String? = null
Operadores contra nulos:

- "?."
- "?:"
- "!!"

Función let
Variables lateinit → ".isInitialized"

Casting de objetos

Palabras reservadas:

- is → Comprobar si una clase es instancia de otra
- as → Forzar el tipo de un objeto a otro (¡cuidado!)
- as? →Casting seguro

Inmutabilidad

Variables

- val → Variable inmutable
- var → Variable mutable

Colecciones

- Collection→ Métodos sólo de consulta
- MutableCollection -> Consulta y métodos de agregación, eliminación, etc.

Iteradores

Rangos → operador in

- in numberFrom..numberTo → Desde el valor indicado hasta el valor final incluído.
- in numberFrom until numberTo → Valor final no incluido.

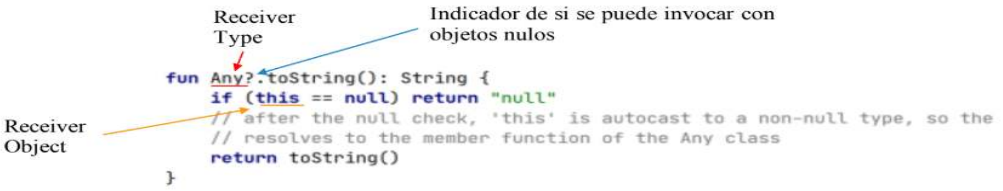
for y while iguales que Java.

Profundizando

Funciones de extensión

Añadir nueva funcionalidad a una clase existente
Se invocan como si fueran métodos de la clase original
Sólo tenemos acceso a métodos y variables públicas
his hace referencia a la propia clase
Se añaden en ficheros y se declaran de forma estática
Se pueden añadir propiedades de extensión

Estructura



Arrays

Proporciona arrays de tipo primitivo como IntArray, ByteArray, BooleanArray, etc
Creación

- Tamaño → val chars = CharArray(4)
- Con valores → val chars = CharArray('H', 'o', 'l', 'a')
- Tamaño y lambda → val ints = IntArray(2) {i -> i*4}

Corrutinas

Definición: Las corutinas permiten realizar programación asíncrona sin bloquear el hilo principal. Parecen secuenciales, pero son asíncronas.

Funcionamiento:

- **Suspend**: Una corrutina puede suspender su ejecución sin bloquear el hilo.
- **Reanudar**: Se reanudan en el punto donde fueron suspendidas.
- **No bloqueantes**: No bloquean el hilo, lo que permite un uso eficiente de los recursos.

Ventajas de usar corutinas

1. **Eficiencia**: No bloquean hilos, permitiendo que otras operaciones sigan ejecutándose.
2. **Código legible**: El código asíncrono es más legible y mantenible en comparación con las cadenas de callbacks.

3. **Compatibilidad con suspensiones**: Facilitan la gestión de operaciones largas con la palabra clave suspend.

Conceptos clave

Funciones suspend

- Son funciones que pueden suspender su ejecución y luego reanudarla.
- Solo pueden ser llamadas desde otras funciones suspend o dentro de corrutinas.

Builders de corutinas

launch: Inicia una corrutina sin devolver un valor.

async: Inicia una corrutina que devuelve un valor a través de un Deferred.

Scopes (Alcances de corutinas)

- **GlobalScope**: Corutinas que duran mientras viva la aplicación (uso peligroso por gestión de ciclo de vida).
- **CoroutineScope**: Permite definir el alcance explícito de una corrutina.
- **viewModelScope**: En Android, las corutinas viven en el ciclo de vida del ViewModel.

Dispatchers

- **Dispatchers.Main**: Para tareas que interactúan con la interfaz de usuario.
- **Dispatchers.IO**: Para operaciones de entrada/salida (I/O), como acceso a archivos o redes.
- **Dispatchers.Default**: Para tareas intensivas en CPU.

Suspender y reanudar (Suspending and Resuming)

- Kotlin permite suspender y reanudar una corrutina con funciones suspend.
- **Ejemplo** del uso de delay para suspender una corrutina sin bloquear el hilo:

Cancelación de corutinas

- Las corutinas pueden ser canceladas en cualquier momento.
- **isActive**: Se usa para verificar si una corutina está activa o ha sido cancelada.