Bloque 1 · Tema 1A

Kotlin

Repaso acelerado del lenguaje para desarrollar en Android

2 horas

Base para AppDummy

RA1



Programación Multimedia y Dispositivos Móviles (0489) · 2º CFGS DAM

Hoja de ruta del tema

7 apartados

1 **Variables**
val frente a var

2 **Null safety**
?. · ?: · !!

3 **Funciones y lambdas**
Parámetros por defecto, it, trailing lambda

4 **Clases**
class · data · sealed · object

5 **Colecciones**
Inmutables y operaciones funcionales

6 **Corrutinas**
suspend, viewModelScope y Flow

7 **Control**
when como expresión y plantillas

¿Qué vamos a evaluar?

RA1

RA1

Aplica tecnologías de desarrollo para dispositivos móviles evaluando sus características y capacidades.

RA1-b	Identificar las tecnologías de desarrollo de aplicaciones para dispositivos móviles
RA1-f	Analizar la estructura de aplicaciones existentes identificando las clases utilizadas
RA1-g	Realizar modificaciones sobre aplicaciones existentes

★ *Kotlin como lenguaje oficial del desarrollo Android*

¿Por qué Kotlin y no Java?

Kotlin

2017

Lenguaje oficial de Android y el recomendado por Google para Jetpack Compose

100 %

Interoperable con Java: cualquier librería Java se usa desde Kotlin sin adaptadores



Menos boilerplate

Sin getters, setters ni código repetitivo.



Seguridad frente a nulos

El sistema de tipos impide el NullPointerException.



Integración nativa con Compose

Lambdas, argumentos con nombre y extensiones.



Corrutinas

Asincronía sin bloquear el hilo principal.

01

Lo esencial

Variables, null safety, funciones y lambdas



val frente a var



val Inmutable

Equivale a final en Java. Una vez asignado, no se reasigna.

```
val nombre: String = "AppDummy"
val version = 1      // infiere Int
// nombre = "Otra"   ✕ no compila
```



var Mutable

Solo cuando el valor tenga que cambiar.

```
var contador = 0
contador = contador + 1   // ☑
```



Regla práctica: declara siempre con val y cambia a var solo cuando la lógica lo exija. Todo es un objeto: no hay tipos primitivos.

Null safety: tres formas de acceder

`String?`

`String → nunca null`

`String? → admite null`

Añadir ? al tipo permite el valor null.
Por defecto, ningún tipo lo acepta.

```
var titulo: String =
    "Inception"
// titulo = null ✕
var descripcion:
    String? = null
```

?.

Llamada segura

Si es null, devuelve null sin fallar

`descripcion?.length`

`→ Int?`

?:

Operador Elvis

Aporta un valor por defecto si es null

`descripcion?.length ?: 0`

`→ Int`

!!

Aserción no nula

Lanza excepción si es null. Evitar

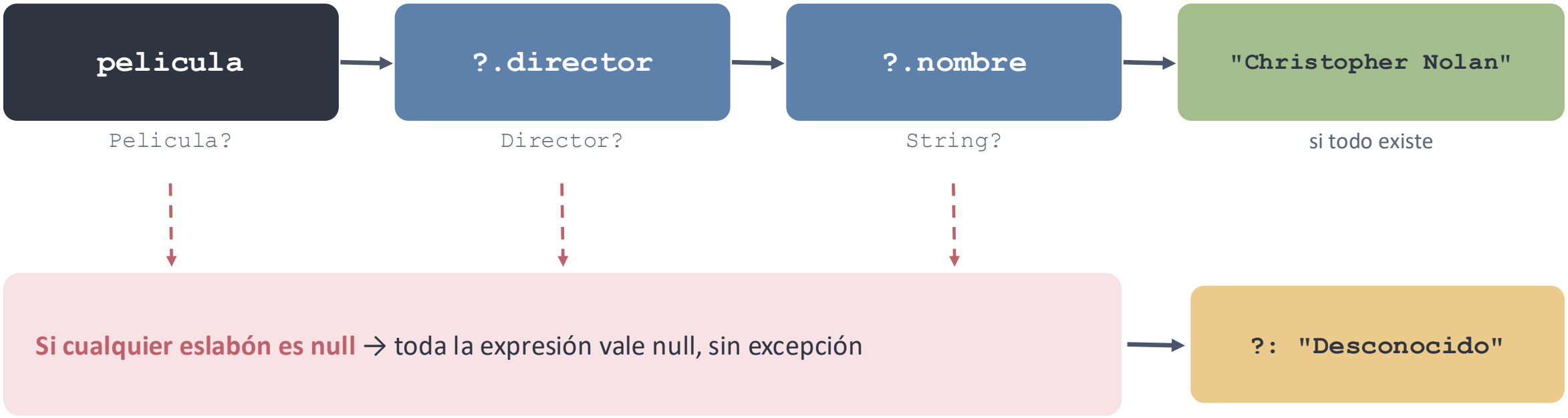
`descripcion!!.length`

`→ Int · ⚠`

Llamadas seguras encadenadas

`?. · ?:`

```
data class Director(val nombre: String?)
data class Pelicula(val titulo: String, val director: Director?)
val nombreDirector = pelicula?.director?.nombre ?: "Desconocido"
```



Anatomía de una función

fun

nombre

parámetro

valor por defecto

tipo de retorno

```
fun crearTitulo(texto: String, mayusculas: Boolean = false): String =
    if (mayusculas) texto.uppercase() else texto
```

cuerpo de expresión: return implícito

```
crearTitulo("appdummy")
```

→ `"appdummy"` Usa el valor por defecto

```
crearTitulo("appdummy", true)
```

→ `"APPDUMMY"` Argumentos posicionales

```
crearTitulo(texto = "appdummy",
            mayusculas = true)
```

→ `"APPDUMMY"` Argumentos con nombre: muy usados en Compose

Lambdas y funciones de orden superior

{ -> }

```
{ numero -> numero * 2 }
```

parámetros · flecha · cuerpo

it Nombre implícito cuando hay un solo parámetro

```
val duplicar: (Int) -> Int = { it * 2 }
```

trailing La última lambda va fuera de los paréntesis

```
aplicar(5) { it + 10 }   // 15
```

Función de orden superior

```
// Recibe una función como parámetro
fun aplicar(
    valor: Int,
    operacion: (Int) -> Int
): Int = operacion(valor)
```

Lo verás en cada Composable

```
Button(onClick = { contador++ }) {
    Text("+")
}

onClick: () -> Unit
```

02

Clases

class · data class · sealed class · object



Cuatro tipos de clase, cuatro usos

Clases



`class`

Clase regular

Constructor primario en la cabecera y bloque init para validar.

Uso típico: Pelicula(titulo, anyo)



`data class`

Modelar datos

equals, hashCode, toString y copy generados.

Uso típico: Modelos de MVVM



`sealed class`

Jerarquía cerrada

El compilador conoce todas las subclases: when sin else.

Uso típico: Estado de la UI



`object`

Singleton

Una única instancia gestionada por Kotlin.

Uso típico: Configuración global

Clase regular: constructor primario e init

class

Pelicula.kt

```
class Pelicula(
    val titulo: String,
    val anyo: Int,
    var puntuacion: Double = 0.0
) {
    init {
        require(anyo > 1888) {
            "El año debe ser posterior a 1888" }
    }
    fun descripcion() =
        "$titulo ($anyo) — ★ $puntuacion"
}
```

1

Constructor primario

Los parámetros val/var son propiedades directamente.

2

Valor por defecto

puntuacion puede omitirse al crear la instancia.

3

Bloque init

Se ejecuta al crear el objeto: ideal para validar.

4

Métodos

Funciones miembro, también con cuerpo de expresión.

```
Pelicula("Dune", 2021, 8.0).descripcion() → Dune (2021) — ★ 8.0
```

data class: el compilador escribe por ti

data class

equals()

hashCode()

```
data class Pelicula(
    val id: Int,
    val titulo: String,
    val puntuacion: Double)
```

toString()

copy()

p1

```
id = 1
titulo = "Dune"
puntuacion = 8.0
```

```
p1.copy(puntuacion = 9.0)
```

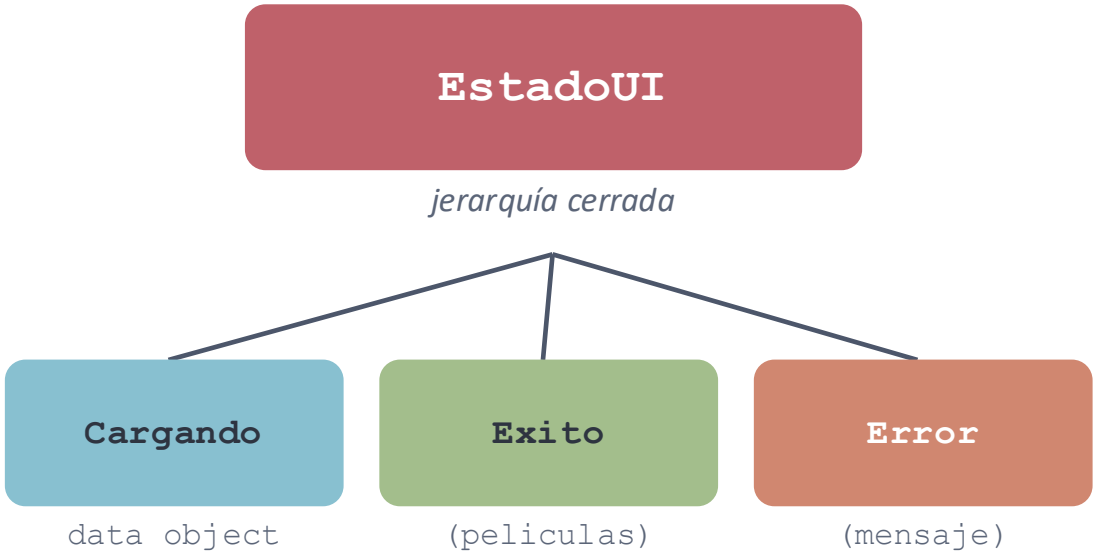
p2

```
id = 1
titulo = "Dune"
puntuacion = 9.0
```

p1 == p2 → **false** compara por valor, no por referencia. copy crea un objeto nuevo: p1 no cambia.

sealed class: estados de la UI sin sorpresas

`sealed class`



when sin else

Si se añade un estado nuevo y no se trata, el código no compila.

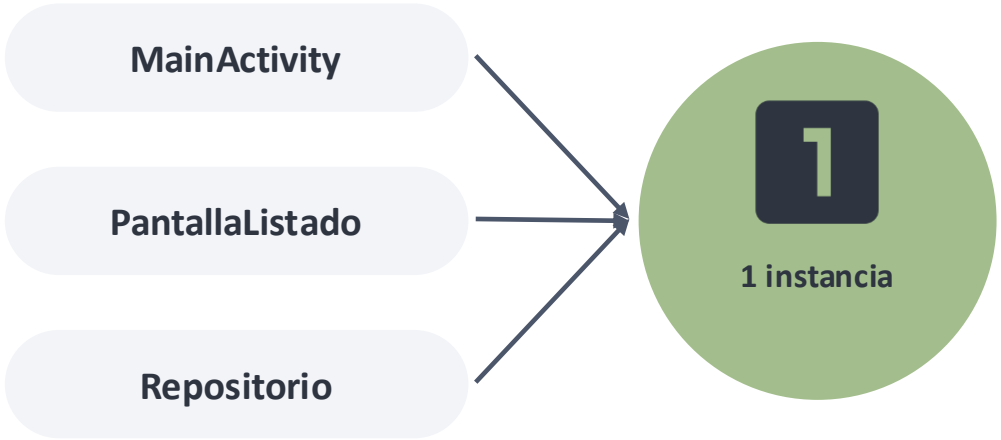
EstadoUI.kt

```kotlin
sealed class EstadoUI {
    data object Cargando : EstadoUI()
    data class Exito(
        val peliculas: List<Pelicula>
    ) : EstadoUI()
    data class Error(
        val mensaje: String
    ) : EstadoUI()
}

when (estado) {
    is EstadoUI.Cargando -> "Cargando..."
    is EstadoUI.Exito -> "${estado.peliculas.size}"
    is EstadoUI.Error -> estado.mensaje
}
```

object y companion object

Singleton



companion object

Equivale a los miembros static de Java, dentro de una clase.

```
object
object ConfiguracionApp {
    const val VERSION = "1.0"
    const val NOMBRE = "AppDummy"
    fun mostrarInfo() =
        println("$NOMBRE v$VERSION")
}
```

```
companion object
class ConexionBD private constructor() {
    companion object {
        const val NOMBRE_BD = "appdummy_db"
        // factory method
        fun crear() = ConexionBD()
    }
}
val bd = ConexionBD.crear()
```

03

Colecciones, corrutinas y control

Operaciones funcionales, suspend y when



Colecciones inmutables y mutables

Tipo	Inmutable (preferida)	Mutable
List	`listOf(...)`	`mutableListOf(...)`
Map	`mapOf(k to v)`	`mutableMapOf(...)`
Set	`setOf(...)`	`mutableSetOf(...)`

```
val generos =
  mapOf("Acción" to 5,
        "Drama" to 3)

val favoritas =
  mutableListOf("Dune")
favoritas.add("Interstellar")
```



Inmutable por defecto

No se añade ni elimina nada: para cambiar se crea una lista nueva. Es lo que espera Compose para detectar cambios.

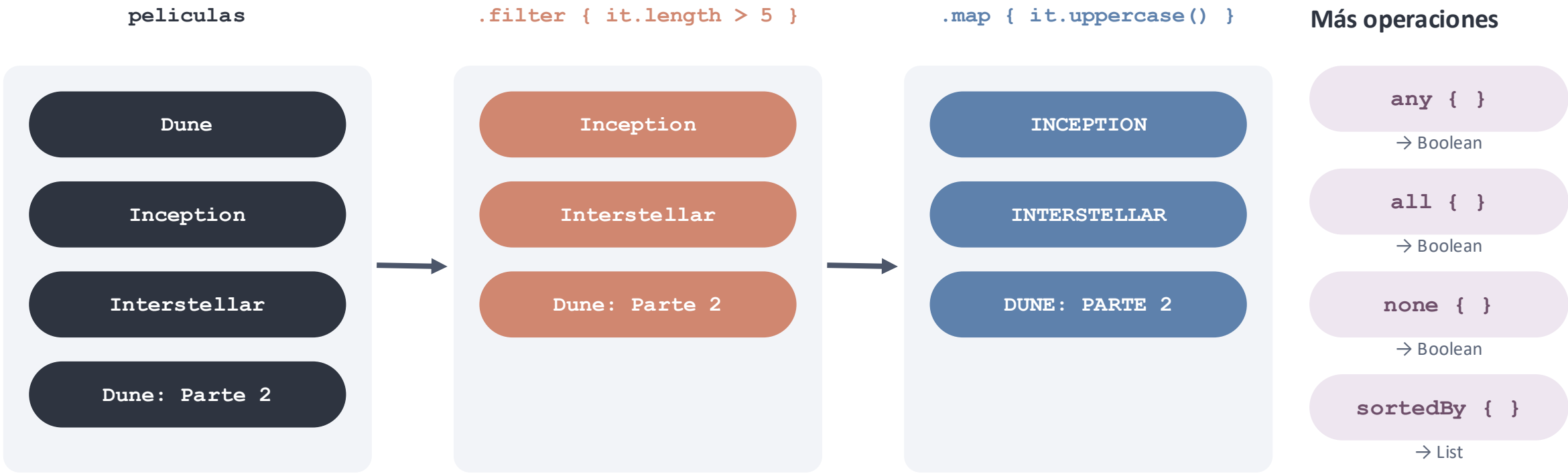


Mutable solo si hace falta

add, remove y modificación en el sitio. Útil dentro de una función, no como estado de la UI.

Operaciones funcionales: una tubería de datos

`filter · map`




```
val titulosLargos = peliculas.filter { it.length > 5 }.map { it.uppercase() }
val hayDune = peliculas.any { it.contains("Dune") }   // true
```

Corrutinas: esperar sin bloquear

suspend

Hilo principal (UI)

La UI sigue respondiendo: dibuja, recibe toques…

Corrutina

llamada	suspendida · delay(1000)	reanuda

```
suspend fun cargarPeliculas(): List<String> {
    delay(1000)   // espera de red simulada
    return listOf("Dune", "Inception")
}
// Solo se llama desde una corrutina u otra
// función suspend (runBlocking: solo en pruebas)
```

suspend fun

Función que se pausa sin bloquear el hilo

Lo verás en: Room · Retrofit2

viewModelScope

Ámbito ligado al ciclo de vida del ViewModel

Lo verás en: Bloque 2

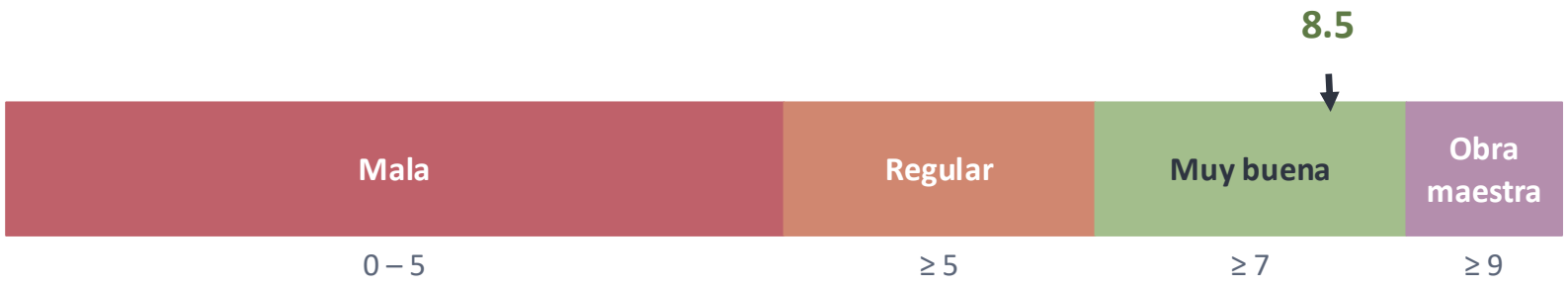
Flow<T>

Flujo reactivo de datos

Lo verás en: Room · Bloque 3

when como expresión y plantillas de cadena

when




```kotlin
val puntuacion = 8.5
val clasificacion = when {
    puntuacion >= 9.0 -> "Obra maestra"
    puntuacion >= 7.0 -> "Muy buena"
    puntuacion >= 5.0 -> "Regular"
    else              -> "Mala"
}   // "Muy buena": se evalúa de arriba abajo
```

Más que un switch

Devuelve un valor, acepta condiciones arbitrarias y tipos (is) con smart cast.

String templates

```kotlin
"$titulo ($anyo)"
"${titulo.uppercase()}"
```

`$var` · `${expresión}`

Ideas clave



val por defecto

var solo cuando el valor deba cambiar.



Null safety

String? + ?. y ?: en lugar de !!.



Lambdas

onClick: () -> Unit y trailing lambda.



data / sealed class

Modelos de datos y estados de la UI.



Colecciones inmutables

filter, map y sortedBy crean listas nuevas.



suspend y Flow

Asincronía sin bloquear el hilo principal.

Referencias

Fuentes



Kotlin — Documentación oficial
kotlinlang.org/docs/home.html



Kotlin para desarrolladores Android
developer.android.com/kotlin/learn



Kotlin Koans — ejercicios interactivos
play.kotlinlang.org/koans



Corrutinas de Kotlin en Android
developer.android.com/kotlin/coroutines



Tema 1A. Kotlin para Android (apuntes)
documentation.javiercarrasco.es/pmdm/B1/B1-T1a_Kotlin_Repaso



Anexo B1-A1 — Kotlin referencia completa
documentation.javiercarrasco.es/pmdm/B1/B1-A1_Kotlin_Referencia_Anexo