Kotlin Fundamentals – Kevin Jones

Este curso introduce los fundamentos del lenguaje Kotlin, destacando su sintaxis moderna, seguridad ante errores comunes y aplicación práctica en el desarrollo de aplicaciones Android y de servidor.

Tema 1: Variables y Tipos

```
fun main() {
    val nombre = "Roberto"
    var edad = 22
    println("Nombre: $nombre, Edad: $edad")
}
```

Resumen:

- `val`: para valores que no cambian.
- `var`: para valores que pueden modificarse.

Tema 2: Control de Flujo

```
fun main() {
    val calificacion = 85
    when {
        calificacion >= 90 -> println("Excelente")
        calificacion >= 70 -> println("Bien")
        else -> println("Requiere mejora")
    }
}
```

Resumen:

- `when` actúa como un `switch`.
- Control de decisiones con `if`, `else`, `when`.

Tema 3: Funciones y Lambdas

```
fun cuadrado(x: Int): Int = x * x

fun main() {
    val numeros = listOf(2, 3, 4)
    val resultado = numeros.map { cuadrado(it) }
    println(resultado)
}
```

Resumen:

- Las funciones pueden declararse con o sin `return`.
- Las lambdas permiten operaciones funcionales.

Tema 4: Clases y Objetos

```
data class Persona(val nombre: String, val edad: Int)

fun main() {
    val persona = Persona("Roberto", 22)
    println(persona)
}
```

Resumen:

- `data class` es ideal para representar datos simples.
- Se crean instancias fácilmente y se imprimen automáticamente.

Tema 5: Colecciones

```
fun main() {
    val lista = listOf("Ana", "Luis", "Roberto")
    val filtrado = lista.filter { it.startsWith("R") }
    println(filtrado)
}
```

Resumen:

- `filter`, `map`, `forEach` permiten trabajar con listas de forma funcional.

Tema 6: Null Safety

```
fun main() {
    val nombre: String? = null
    println(nombre?.length ?: "No hay nombre")
}
```

Resumen:

- Kotlin evita errores de `null` con `?`, `?:` y `!!`.

Tema 7: Corutinas (Intro)

```
import kotlinx.coroutines.*

fun main() = runBlocking {
    launch {
        delay(1000)
        println("Desde la corrutina")
    }
    println("Inicio")
}
```

Resumen:

- `launch` inicia una tarea en segundo plano.
- `delay` simula espera sin bloquear la ejecución principal.

Programación Orientada a Objetos en Kotlin – Kevin Jones

Este curso profundiza en los principios de programación orientada a objetos con Kotlin, usando un enfoque práctico.

Tema 1: Clases

```
class Persona(val nombre: String, var edad: Int) {
    fun saludar() = println("Hola, soy $nombre y tengo $edad años")
}

fun main() {
    val p = Persona("Roberto", 22)
    p.saludar()
}
```

Resumen:

- Las clases agrupan datos y comportamientos.
- Se definen atributos y métodos.

Tema 2: Herencia

```
open class Animal(val nombre: String) {
    open fun hablar() = println("$nombre hace un sonido")
}

class Perro(nombre: String): Animal(nombre) {
    override fun hablar() = println("$nombre dice Guau!")
}

fun main() {
    val perro = Perro("Firulais")
    perro.hablar()
}
```

Resumen:

- Las clases `open` permiten herencia.
- `override` redefine métodos en subclases.

Tema 3: Interfaces

```
interface Volador {
    fun volar()
}
```

```
class Pajaro : Volador {
    override fun volar() = println("El pájaro vuela")
}

fun main() {
    val p = Pajaro()
    p.volar()
}
```

Resumen:

- Las interfaces definen comportamientos sin implementación concreta.

Tema 4: Data Classes

```
data class Libro(val titulo: String, val autor: String)

fun main() {
    val libro = Libro("Kotlin", "Kevin Jones")
    println(libro)
}
```

Resumen:

- Las `data class` generan automáticamente métodos útiles para datos inmutables.

Tema 5: Objetos (Singleton)

```
object Configuracion {
    val appName = "Mi App"
    fun mostrar() = println("App: $appName")
}

fun main() {
    Configuracion.mostrar()
}
```

Resumen:

- `object` crea una única instancia global, útil para configuraciones.

Tema 6: Enum y Clases Selladas

```
enum class Dia { LUNES, MARTES, MIERCOLES }

sealed class Resultado
class Exito(val dato: String) : Resultado()
class Error(val mensaje: String) : Resultado()

fun manejar(resultado: Resultado) {
    when (resultado) {
        is Exito -> println("OK: ${resultado.dato}")
        is Error -> println("Fallo: ${resultado.mensaje}")
    }
}

fun main() {
    manejar(Exito("Completado"))
    manejar(Error("Fallo de red"))
}
```

Resumen:

- `enum`: valores constantes.
- `sealed class`: jerarquías seguras para representar estados con control exhaustivo.