📘 Actividades de Verificación – Unidad 4: Componentes de la arquitectura (Android Basics with Compose)

# 🎯 Objetivo general

Comprobar que:  
- Has realizado el codelab y visto los videos de la Unidad 4.  
- Comprendes los conceptos de ciclo de vida, ViewModel, estado y pruebas unitarias.  
- Puedes aplicar lo aprendido en una pequeña app propia.

# 1. Cuestionario de repaso (10 min)

Instrucciones: Responde de manera individual. Algunas preguntas son de selección múltiple, otras de respuesta breve.

* 1. ¿Qué sucede con el ciclo de vida de la Activity al rotar el dispositivo?  
   a) Se mantiene igual, no cambia.  
   b) Se destruye y se vuelve a crear.  
   c) Solo se reinicia el método onResume().
* 2. Verdadero o falso: Un ViewModel sobrevive a los cambios de configuración (como la rotación de pantalla).
* VERDADERO
* 3. Explica con tus propias palabras: ¿Por qué no es recomendable guardar el estado directamente en la Activity?

Porque se pierde todo al rotar. Activity muere y renace al rotar, viewmodel sobrevive; respecto al estado

* 4. ¿Qué tipo de lógica se recomienda ubicar dentro de un ViewModel?  
   a) Lógica de UI.  
   b) Lógica de negocio y manejo de estado.  
   c) Recursos de diseño (colores, estilos).
* 5. En el codelab de pruebas unitarias, ¿qué parte del ViewModel se pone a prueba y por qué?

Se pone a prueba la logica y la inteligencia de la app, el manejo del estado

# 2. Ejercicio de completar código (15 min)

Instrucciones: Completa los huecos en el siguiente código.

class CounterViewModel : ViewModel() {  
 private val \_count = MutableStateFlow(0)  
 val count: StateFlow<Int> = \_count.\_\_\_\_() // completar  
  
 fun increment() {  
 \_count.\_\_\_\_ { old ->  
 old + 1  
 }  
 }  
}

@Composable  
fun CounterScreen(viewModel: CounterViewModel = viewModel()) {  
 val value by viewModel.count.collectAsState()  
  
 Column {  
 Text(text = "Valor: $value")  
 Button(onClick = { viewModel.\_\_\_\_() }) { // completar  
 Text("Incrementar")  
 }  
 }  
}

Pregunta extra: ¿Qué pasará si giras el dispositivo? ¿Por qué?

class CounterViewModel : ViewModel() {

private val \_count = MutableStateFlow(0)

val count: StateFlow<Int> = \_count.asStateFlow()

fun increment() {

\_count.update { old ->

old + 1

}

}

}

@Composable

fun CounterScreen(viewModel: CounterViewModel = viewModel()) {

val value by viewModel.count.collectAsState()

Column {

Text(text = "Valor: $value")

Button(onClick = { viewModel.increment() }) {

Text("Incrementar")

}

}

}

Si se gira el dispositivo, el contador se mantiene por estar usando viewmodel.

# 3. Reto práctico: extensión del codelab (2 horas para 1 de octubre)

Enunciado:  
Crea una pequeña aplicación Compose que tenga:  
- Un contador controlado por un ViewModel.  
- Un botón para reiniciar el contador a 0.  
- Prueba unitaria que verifique el correcto funcionamiento de la función reset().  
- Evidencia de que al rotar el dispositivo el valor no se pierde.

Entrega:  
- Código fuente en un repositorio o ZIP.  
- Captura de pantalla antes y después de la rotación.  
- Resultado de las pruebas unitarias (pantallazo o log).