

**Sesión 01:**  
**Xcode, Swift, SwiftUI e iOS: Introducción**  
Grupo de Trabajo  
Líneas de Productos Software

Pablo Gómez Rivas

Departamento de Informática  
Universidad de Almería

Almería, martes 24 de septiembre de 2024

# Índice de Contenidos

Página

Introducción al lenguaje Swift .....	3
¿Se mejora la legibilidad si hacemos uso de interpolación en lugar de concatenación de cadenas? .....	3
¿Qué tenemos que hacer si queremos trabajar con números decimales? .....	3
¿Qué ocurre con el orden al mostrar el contenido de un diccionario? .....	3
¿Qué diferencias existen entre los componentes Text y Label? .....	4
¿Qué sentido tiene que el botón esté activo si el campo de texto está vacío? .....	4
Ejercicio 01 .....	5
Ejercicio 02 .....	6
Ejercicio 03 .....	7
Ejercicio 04 .....	8
Ejercicio 05 .....	9
Desarrollo de una app: Interfaz de usuario .....	9
Paso 1 .....	9
Paso 2 .....	10
Paso 3 .....	11
Paso 4 .....	12
Paso 5 .....	12
Paso 6 .....	13
Paso 7 .....	13
Paso 8 .....	14
Paso 9 .....	14
Paso 10 y Ejercicio 6 .....	15
SaludoText.swift .....	17

## Introducción al lenguaje Swift

### ¿Se mejora la legibilidad si hacemos uso de interpolación en lugar de concatenación de cadenas?

```
var str: String = String(num1) + " + " + String(num2) + " = " + String(num1 + num2)
```

Este código usa concatenación manual de cadenas, lo que puede volverse tedioso y difícil de leer a medida que aumenta la complejidad.

En lugar de convertir manualmente cada número a cadena, la interpolación nos permite integrar las variables directamente dentro del texto. **Sí, se mejora la legibilidad**, ya que la interpolación es más directa, más limpia y reduce la posibilidad de errores.

```
var str: String = "\(num1) + \(num2) = \(num1 + num2)"
```

### ¿Qué tenemos que hacer si queremos trabajar con números decimales?

Para trabajar con números decimales en Swift, necesitamos usar tipos de datos como Double o Float. Si queremos formatear los números decimales para que muestren una cantidad específica de decimales, podemos usar el método String(format:):

```
import UIKit

var num1: Double = 5.1234
var num2: Double = 12.5678

var str: String = String(format: "%.2f", num1 + num2)
print(str) // Imprimirá el resultado con 2 decimales: 17.69
```

#### Tipos numéricos en Swift:

- Int: Entero con tamaño dependiente de la plataforma (32 o 64 bits).
- Int8, Int16, Int32, Int64: Enteros con tamaños de 8, 16, 32, y 64 bits.
- UInt: Entero sin signo (solo positivos).
- Float: Número de punto flotante de 32 bits (menos precisión).
- Double: Número de punto flotante de 64 bits (mayor precisión).

### ¿Qué ocurre con el orden al mostrar el contenido de un diccionario?

En Swift, los diccionarios no garantizan un orden específico para los elementos. No son una estructura ordenada por defecto. Sin embargo, se puede recorrer el diccionario ordenando primero las claves.

#### Iterar sobre el diccionario ordenado por clave:

```
// Imprimir diccionario ordenado por clave
print(diccionario.sorted(by: {$0.key < $1.key}))

// Iterar sobre el diccionario ordenado por clave
for par in diccionario.sorted(by: {$0.key < $1.key}) {
    print("\(par.key) --> \(par.value)")
}
```

El método `sorted(by:)` ordena el diccionario por las claves en orden alfabético (en el caso de las cadenas). En este caso, primero se imprimirá el diccionario ordenado, y luego cada clave y su valor asociado se mostrarán de manera ordenada.

### ¿Qué diferencias existen entre los componentes **Text** y **Label**?

- **Text:** Es un componente utilizado para mostrar una cadena de texto simple en la interfaz de usuario. Sirve para representar contenido visual, como etiquetas, sin una relación específica con datos dinámicos o contexto semántico. No tiene propiedades avanzadas de accesibilidad o vínculo con datos.
- **Label:** Aunque en algunas interfaces no existe un componente específico llamado "Label" (en SwiftUI, por ejemplo, no es comúnmente usado de esa forma), cuando se habla de Label generalmente se refiere a un componente que combina texto con un ícono o que está relacionado con accesibilidad. En otras interfaces, los componentes Label se utilizan para etiquetar elementos como campos de texto o botones, proporcionando contexto adicional o describiendo la función de otro componente de la interfaz.

### ¿Qué sentido tiene que el botón esté activo si el campo de texto está vacío?

Es lógico que un botón de "Reset" no esté activo si no hay nada que borrar. Tener un botón de este tipo activo cuando el campo de texto está vacío no tendría sentido práctico, ya que no se puede "resetear" algo que ya está vacío. Para controlar esta lógica, se puede utilizar el modificador **disabled** en SwiftUI o frameworks similares. Este modificador permite activar o desactivar un botón en función de una condición.

## Ejercicio 01

Probad con distintas funciones matemáticas, diseñando un ejemplo guiado en el que se muestre una funcionalidad determinada (por ejemplo, redondeo de números decimales aleatorios). Haced uso de bucles.

```

Ejercicio1
Ejercicios1-5 ) Ejercicio1 ) No Selection

1 import Foundation
2
3 // Ejercicio 1: Mostramos 4 números aleatorios (entre 0 y 9.9999) mostrando por consola la salida formateada con 4 decimales
4 for i in 0...3 {
5     let number = Double.random(in: 0..<10)
6     print("\(i) .- " + String(format: "%.4f", number)) // Formato con 4 decimales
7 }
8
9 // Generamos 4 números aleatorios, los redondeamos, y mostramos el resultado
10 for i in 0...3 {
11     let number = Double.random(in: 0..<10)
12     let roundedNumber = round(number * 100) / 100 // Redondeamos a 2 decimales
13     print("\(i) .- " + String(format: "%.2f", roundedNumber)) // Formato con 2 decimales
14 }
15
16 // Recorremos un rango de números y determinamos si son pares
17 for i in -5..<10 {
18     if i % 2 == 0 {
19         print("\(i) es par")
20     }
21 }
22
23 // Sintaxis más sencilla con la cláusula "where"
24 for i in -5..<10 where i % 2 == 0 {
25     print("\(i) es par")
26 }

```

```

0 .- 4.4366
1 .- 6.2829
2 .- 5.0313
3 .- 9.5280
0 .- 4.27
1 .- 3.39
2 .- 6.85
3 .- 8.21
-4 es par
-2 es par
0 es par
2 es par
4 es par
6 es par
8 es par
-4 es par
-2 es par
0 es par
2 es par
4 es par
6 es par
8 es par

```

## Ejercicio 02

Vamos a extender el ejemplo anterior permitiendo que cada alumno tenga asociado, no una única nota, sino un array de notas. Vamos a añadir nuevas entradas al diccionario y también añadiremos nuevas notas a las entradas (puede que ya existieran, o no). Cada alumno tendrá un número de notas diferente. Vamos a partir del código que se muestra a continuación:

<

>

Ejercicio2

Ejercicios1-5

Ejercicio2

No Selection

```

1 import Foundation
2
3 // Ejercicio 2: Diccionario de alumnos con arrays de notas
4 var diccionario:[String:[Int]] = [:]
5
6 // Añadiendo algunas entradas iniciales
7 diccionario["andres"] = [2, 4]
8 diccionario["antonio"] = [3, 5, 6, 7]
9 diccionario["juan"] = [1, 4, 3]
10
11 // Añadiendo una nueva entrada o actualizando la existente
12 @MainActor
13 func agregarNotas(alumno: String, nuevasNotas: [Int]) {
14     if var notasExistentes = diccionario[alumno] {
15         // Si ya existen notas para el alumno, agregamos las nuevas
16         notasExistentes.append(contentsOf: nuevasNotas)
17         diccionario[alumno] = notasExistentes
18     } else {
19         // Si no existe el alumno, lo creamos con las nuevas notas
20         diccionario[alumno] = nuevasNotas
21     }
22 }
23
24 // Agregar notas a alumnos existentes o nuevos
25 agregarNotas(alumno: "andres", nuevasNotas: [5, 6])
26 agregarNotas(alumno: "emilia", nuevasNotas: [5, 7])
27 agregarNotas(alumno: "antonio", nuevasNotas: [8])
28
29 // Imprimir el diccionario
30 print(diccionario)
31
32 // Iterar sobre el diccionario ordenado por clave
33 for par in diccionario.sorted(by: { $0.key < $1.key }) {
34     print("\(par.key) --> \(par.value)")
35 }

```

```

["emilia": [5, 7], "andres": [2, 4, 5, 6], "juan": [1, 4, 3], "antonio": [3, 5, 6, 7, 8]]
andres --> [2, 4, 5, 6]
antonio --> [3, 5, 6, 7, 8]
emilia --> [5, 7]
juan --> [1, 4, 3]

```



## Ejercicio 04

Y ya, para terminar esta parte, vamos a crear una función que calcule la nota media de un alumno. El diccionario se mostrará ahora en Consola siguiendo el orden descendente según la nota media de cada alumno

```

1 import Foundation
2
3 // Ejercicio 4: Diccionario de alumnos con arrays de notas
4 var diccionario:[String:[Int]] = [:]
5
6 // Añadiendo algunas entradas iniciales
7 diccionario["andres"] = [2, 4]
8 diccionario["antonio"] = [3, 5, 6, 7]
9 diccionario["juan"] = [1, 4, 3]
10
11 // Función para calcular la nota media
12 func notaMedia(notas: [Int]) -> String {
13     let suma = notas.reduce(0, +)
14     let media = Double(suma) / Double(notas.count)
15     return String(format: "%.2f", media)
16 }
17
18 // Función para calcular estadísticas de notas
19 func estadisticasNotas(notas: [Int]) -> (maxima: Int, minima: Int, suma: Int, media: Double) {
20     let maxima = notas.max() ?? 0
21     let minima = notas.min() ?? 0
22     let suma = notas.reduce(0, +)
23     let media = Double(suma) / Double(notas.count)
24     return (maxima, minima, suma, media)
25 }
26
27 // Ordenar el diccionario por nota media en orden descendente
28 let ordenPorNotaMedia = diccionario.sorted {
29     let media1 = Double($0.value.reduce(0, +)) / Double($0.value.count)
30     let media2 = Double($1.value.reduce(0, +)) / Double($1.value.count)
31     return media1 > media2
32 }
33
34 // Mostrar el diccionario ordenado por nota media
35 print("Diccionario ordenado por nota media (descendente):")
36 for par in ordenPorNotaMedia {
37     let media = notaMedia(notas: par.value)
38     print("\(par.key) --> \(media)")
39 }
40
41 // Ejemplo extra: Calcular estadísticas para un array de notas
42 let notasEjemplo = [3, 5, 7, 2, 6]
43 let estadisticas = estadisticasNotas(notas: notasEjemplo)
44 print("\nEstadísticas de notas:")
45 print("Nota máxima: \(estadisticas.maxima)")
46 print("Nota mínima: \(estadisticas.minima)")
47 print("Suma de notas: \(estadisticas.suma)")
48 print("Nota media: \(String(format: "%.2f", estadisticas.media))")

```

■ ▶

Diccionario ordenado por nota media (descendente):  
 antonio --> 5.25  
 andres --> 3.00  
 juan --> 2.67

Estadísticas de notas:  
 Nota máxima: 7  
 Nota mínima: 2  
 Suma de notas: 23  
 Nota media: 4.60



## Ejercicio 05

<

>

Ejercicio5

Ejercicios1-5

Ejercicio5

No Selection

```

1 import Foundation
2
3 // Ejercicio 5: Diccionario de alumnos con arrays de notas
4 var diccionario: [String: [Int]] = [:]
5
6 // Añadiendo algunas entradas iniciales
7 diccionario["andres"] = [2, 4]
8 diccionario["antonio"] = [3, 5, 6, 7]
9 diccionario["juan"] = [1, 4, 3]
10
11 // Función para calcular las estadísticas de notas
12 func estadisticas(notas: [Int]) -> (max: Int, min: Int, suma: Int, med: String) {
13     let maxima = notas.max() ?? 0
14     let minima = notas.min() ?? 0
15     let suma = notas.reduce(0, +)
16     let media = Double(suma) / Double(notas.count)
17     let mediaFormateada = String(format: "%.2f", media)
18     return (max: maxima, min: minima, suma: suma, med: mediaFormateada)
19 }
20
21 // Ordenar el diccionario por nota media en orden descendente
22 let ordenPorNotaMedia = diccionario.sorted {
23     let media1 = Double($0.value.reduce(0, +)) / Double($0.value.count)
24     let media2 = Double($1.value.reduce(0, +)) / Double($1.value.count)
25     return media1 > media2
26 }
27
28 // Mostrar el diccionario ordenado por nota media
29 print("Diccionario ordenado por nota media (descendente):")
30 for par in ordenPorNotaMedia {
31     let stats = estadisticas(notas: par.value)
32     print("\(par.key) --> (max: \(stats.max), min: \(stats.min), suma: \(stats.suma), med: \"\(stats.med)\")")
33 }

```

Diccionario ordenado por nota media (descendente):  
 antonio --> (max: 7, min: 3, suma: 21, med: "5.25")  
 andres --> (max: 4, min: 2, suma: 6, med: "3.00")  
 juan --> (max: 4, min: 1, suma: 8, med: "2.67")

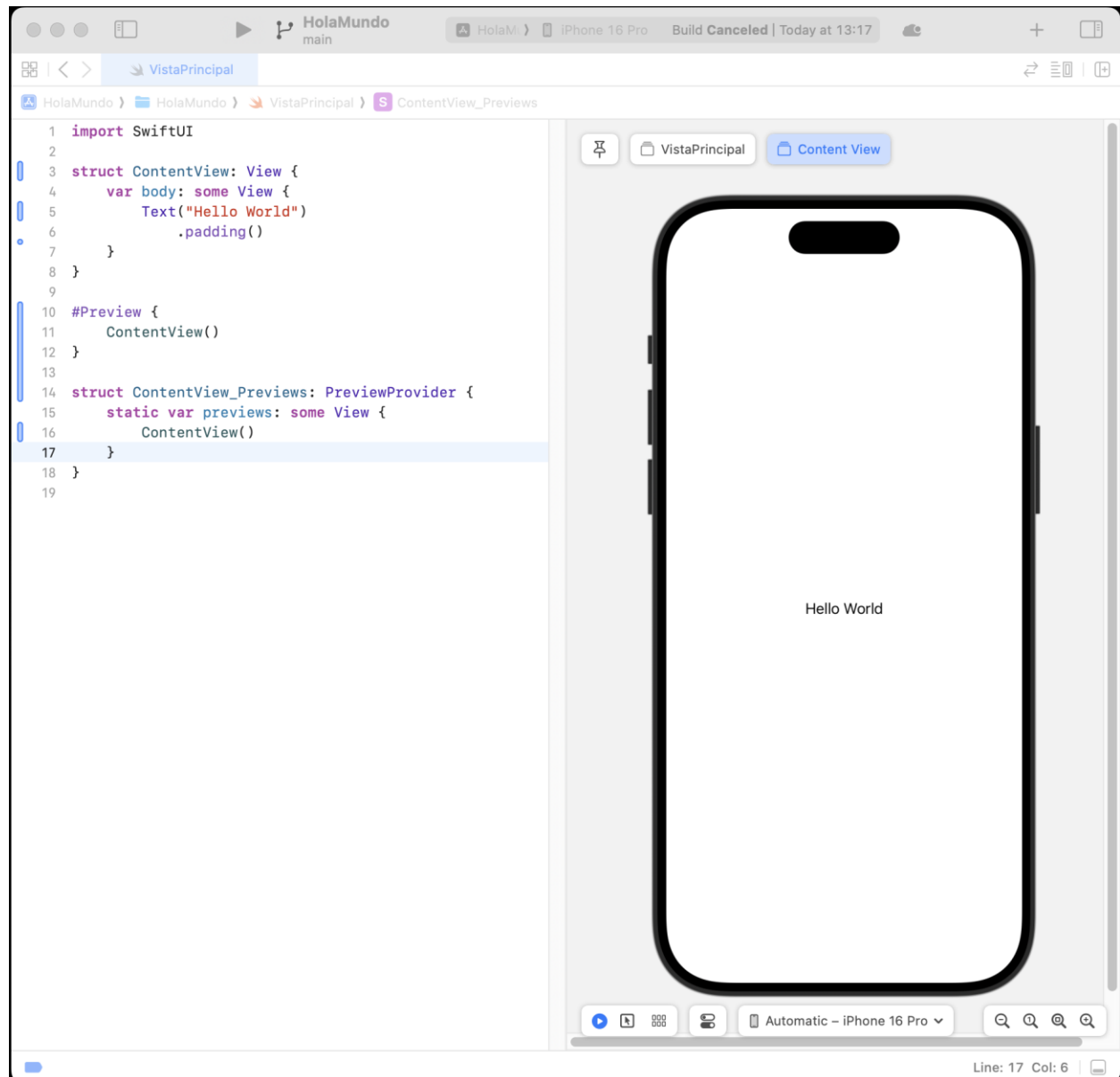
## Desarrollo de una app: Interfaz de usuario

Desarrollamos la app HolaMundo en los siguientes pasos, documentados con capturas:

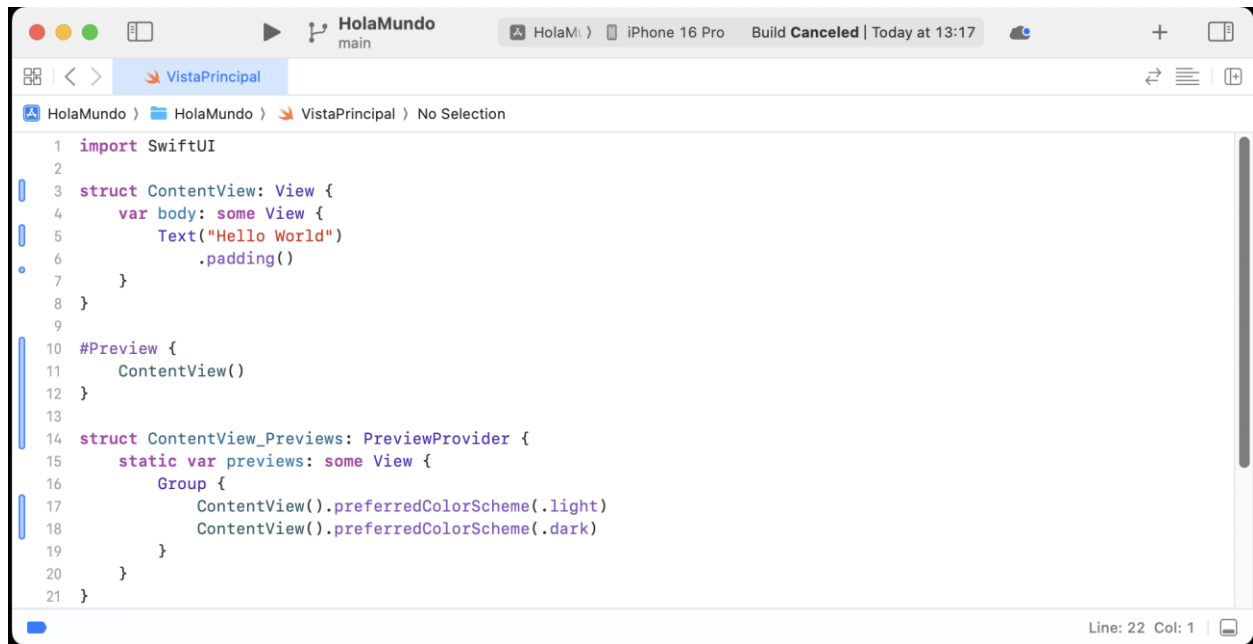
### Paso 1

Creamos un nuevo proyecto Xcode siguiendo los pasos del informe.

## Paso 2



### Paso 3



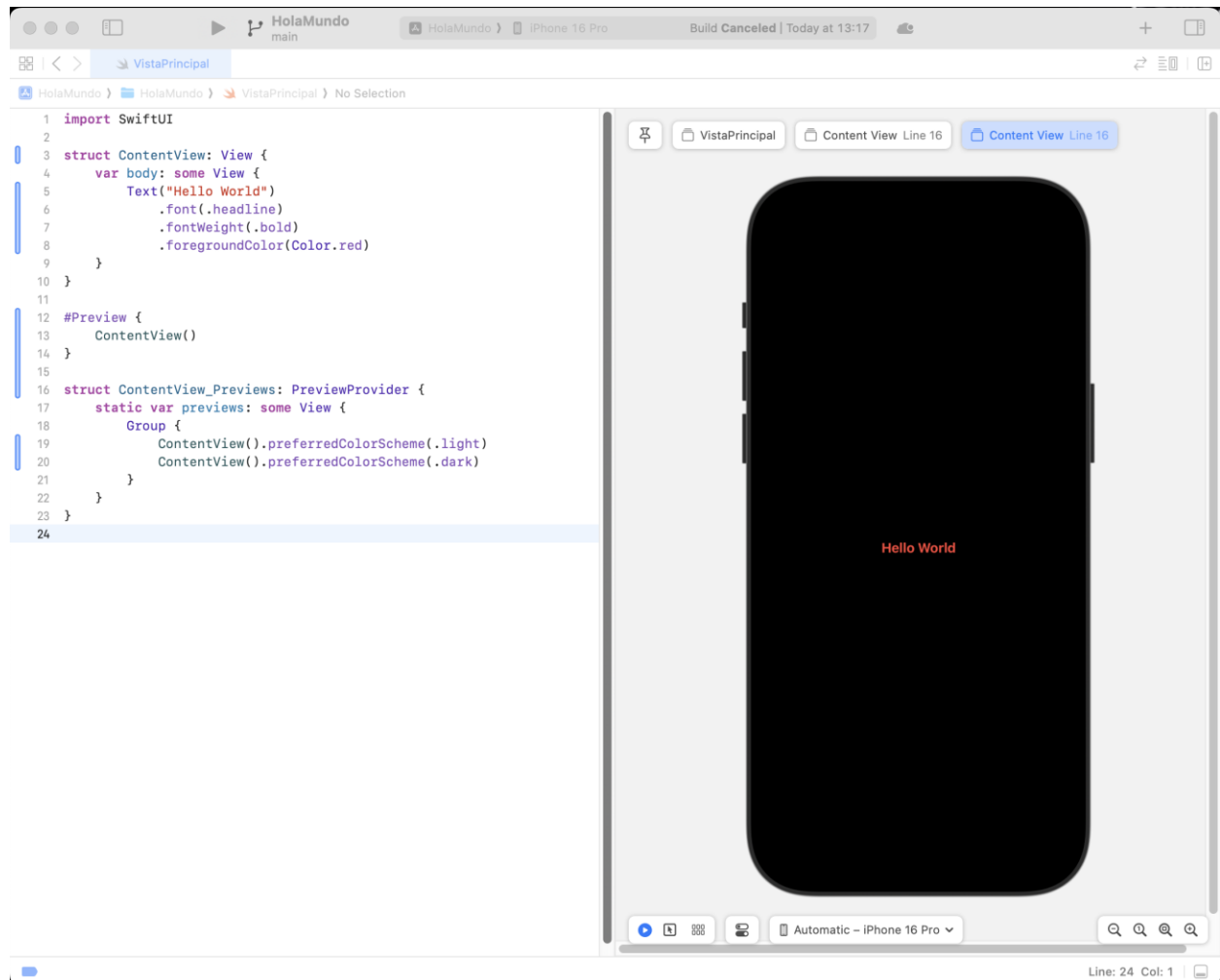
The screenshot shows the Xcode IDE with a project named 'HolaMundo'. The file explorer on the left shows the project structure: 'HolaMundo' (main) containing 'VistaPrincipal'. The code editor displays the following Swift code:

```
1 import SwiftUI
2
3 struct ContentView: View {
4     var body: some View {
5         Text("Hello World")
6         .padding()
7     }
8 }
9
10 #Preview {
11     ContentView()
12 }
13
14 struct ContentView_Previews: PreviewProvider {
15     static var previews: some View {
16         Group {
17             ContentView().preferredColorScheme(.light)
18             ContentView().preferredColorScheme(.dark)
19         }
20     }
21 }
```

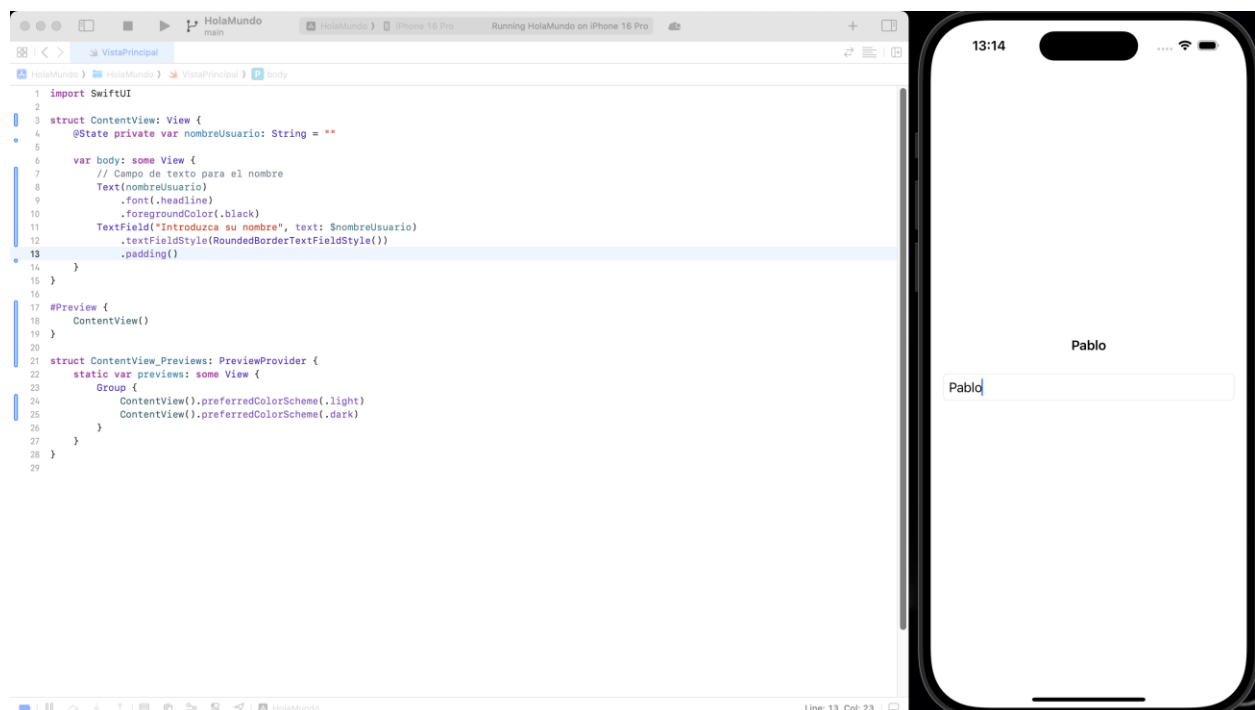
The status bar at the bottom right indicates 'Line: 22 Col: 1'.



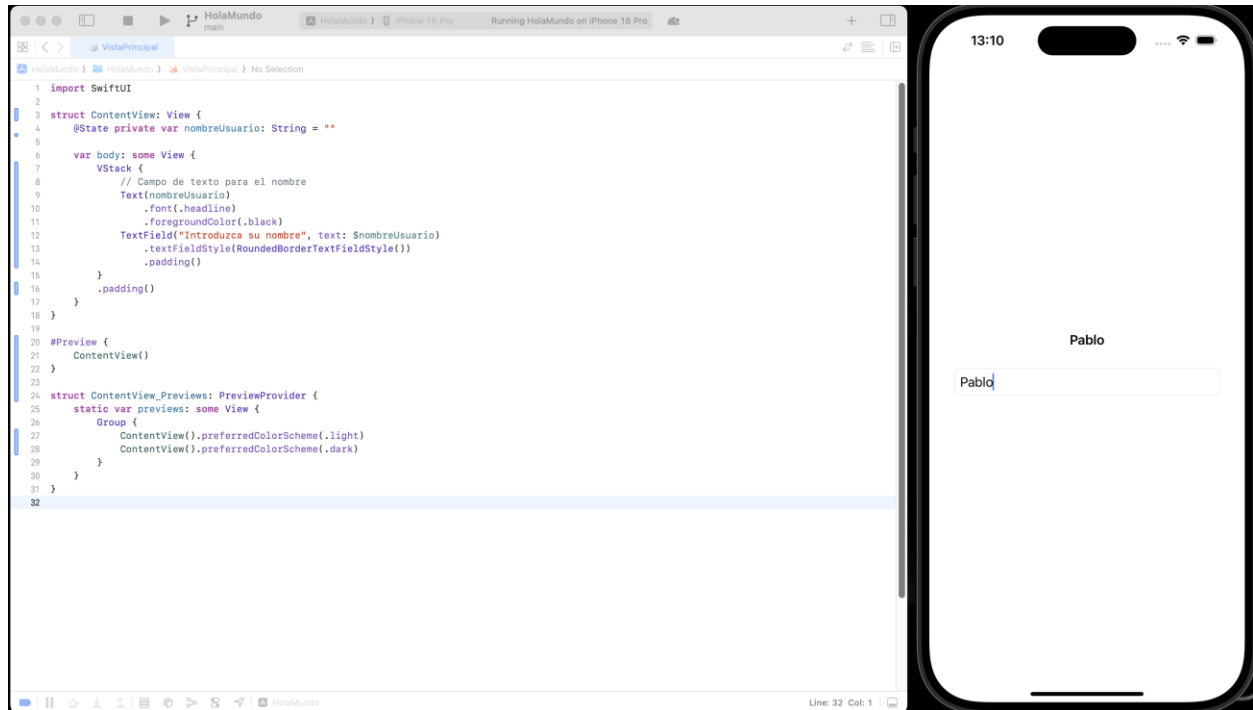
## Paso 4



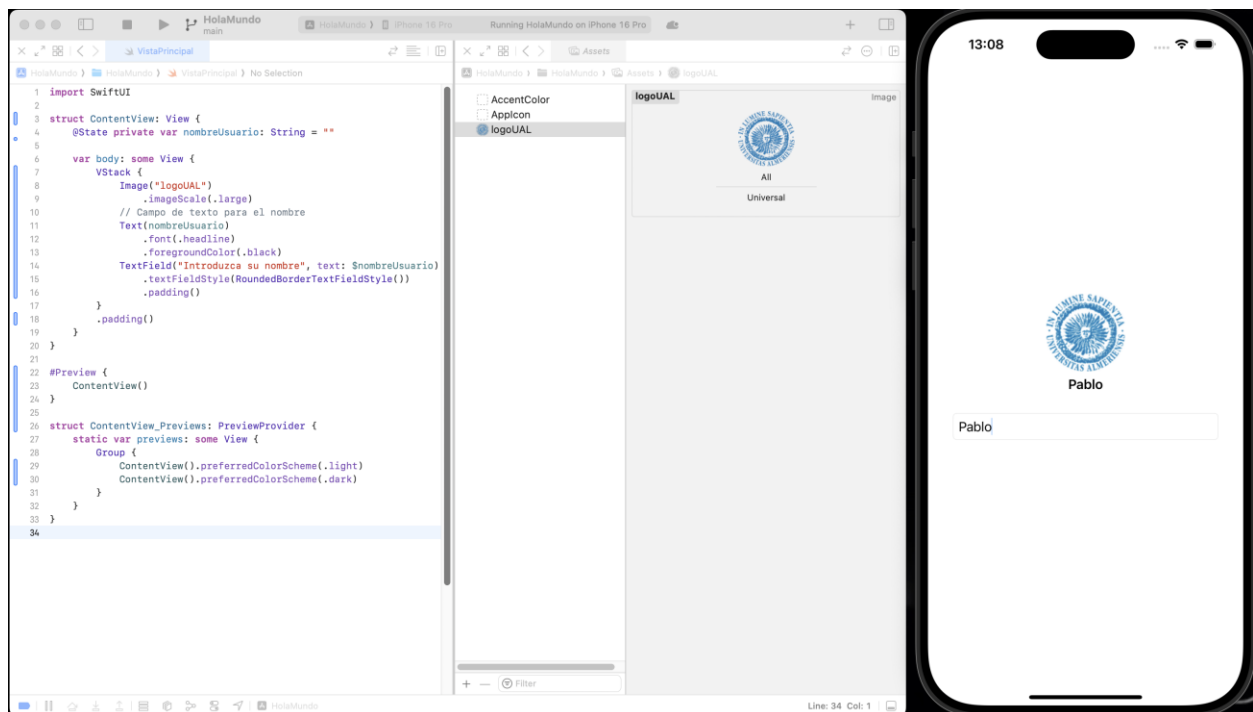
## Paso 5



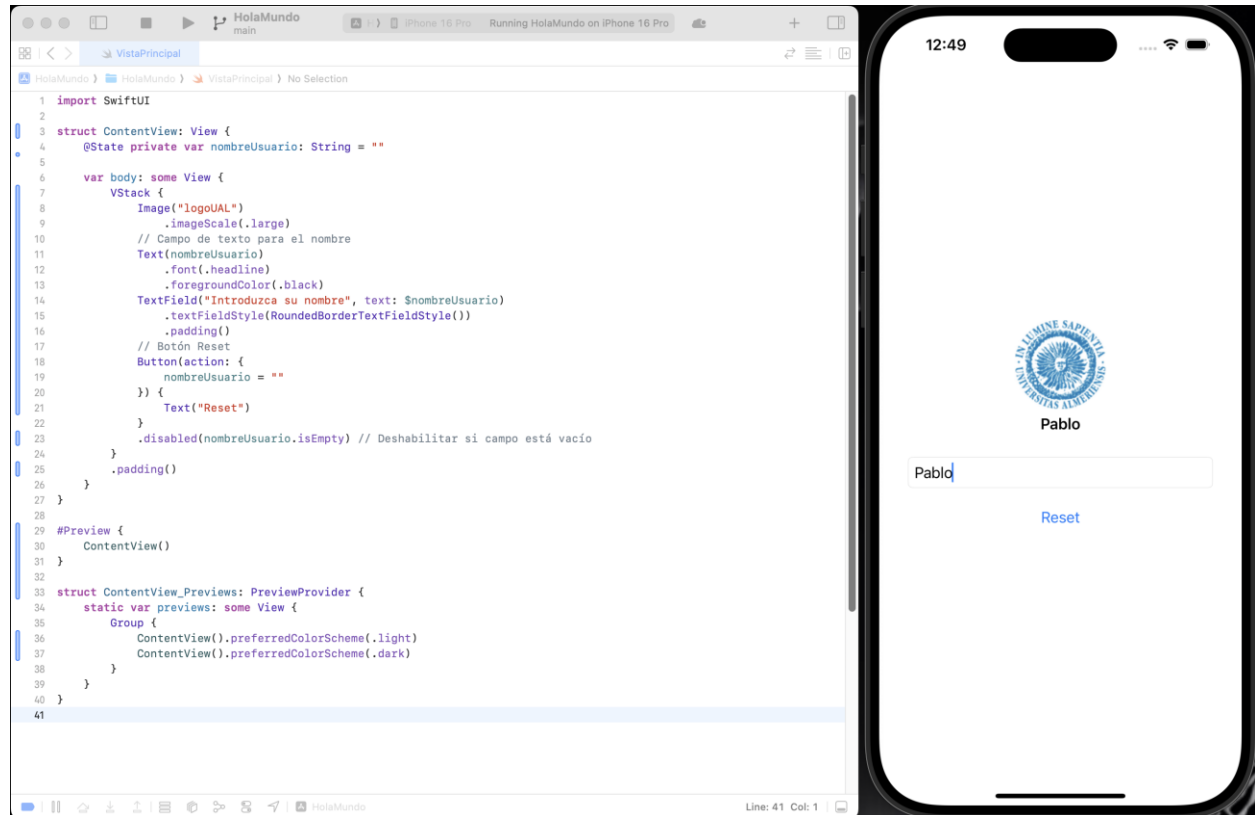
## Paso 6



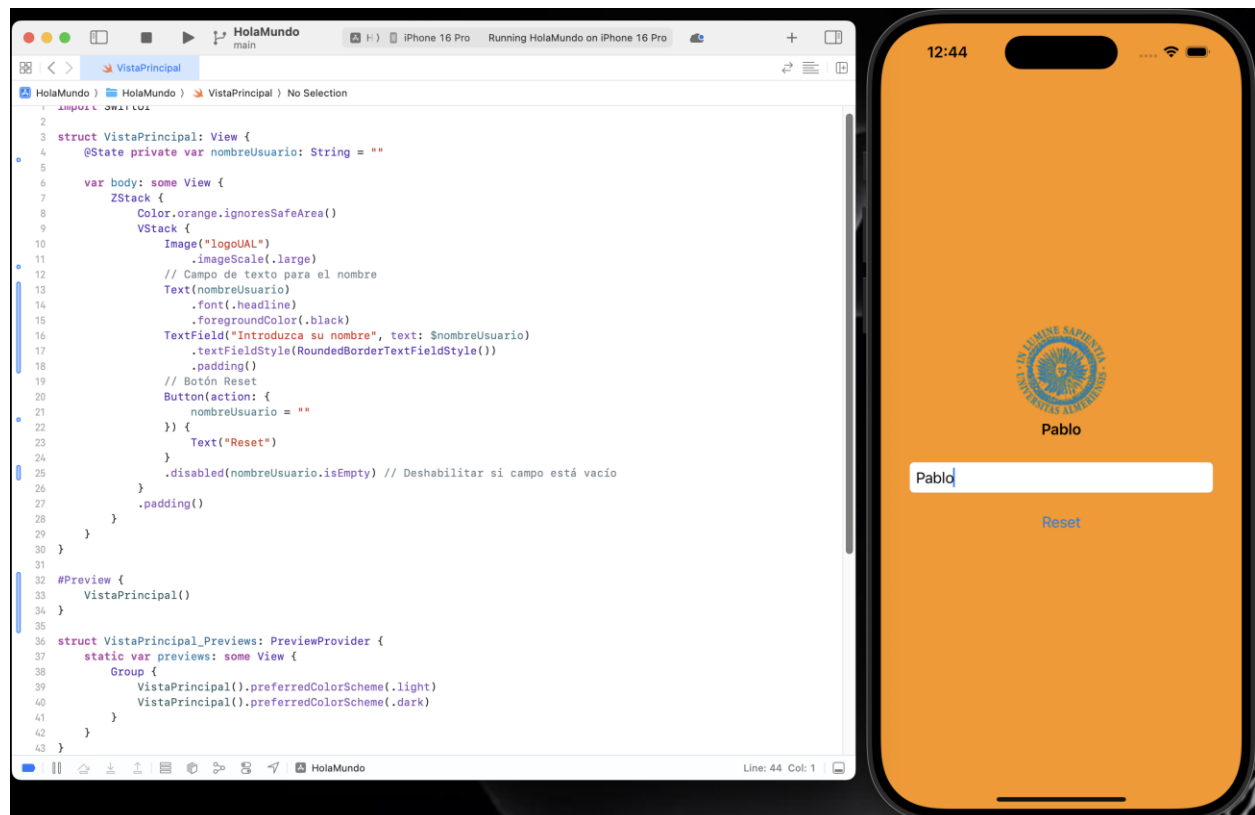
## Paso 7



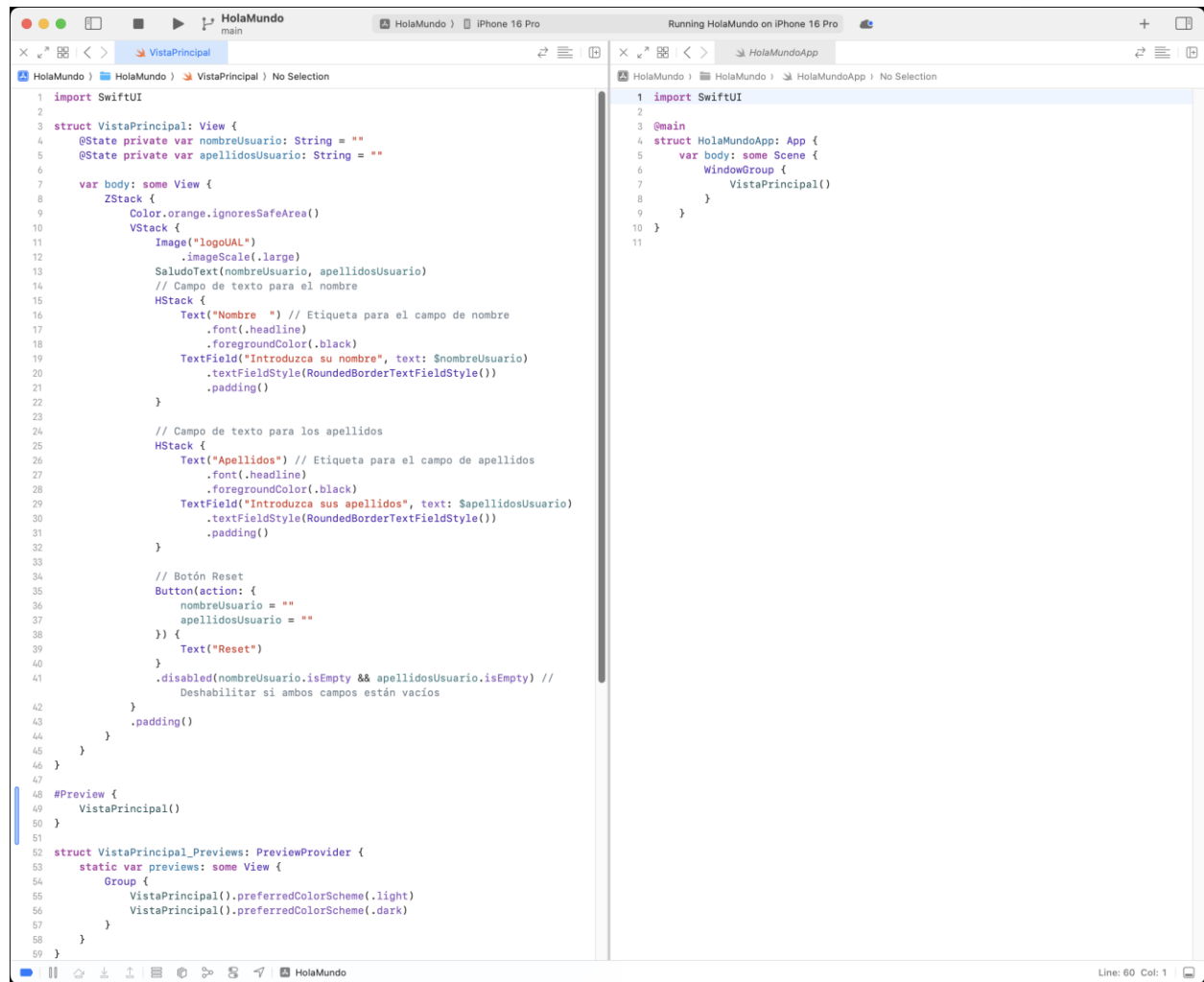
## Paso 8



## Paso 9




## Paso 10 y Ejercicio 6







## SaludoText.swift

 | < > SaludoText HolaMundo >  HolaMundo >  SaludoText > No Selection

```
1  import SwiftUI
2
3  struct SaludoText: View {
4      private var msg: String
5
6      // Inicializador
7      init(_ nombreUsuario: String, _ apellidosUsuario: String) {
8          if nombreUsuario.isEmpty && apellidosUsuario.isEmpty {
9              msg = "Hola, ¿qué tal?"
10             } else {
11                 msg = "Hola, \(nombreUsuario + " " + apellidosUsuario)"
12             }
13     }
14
15     var body: some View {
16         Text(msg)
17             .font(.title)
18             .fontWeight(.black)
19             .foregroundColor(Color.blue)
20             .padding()
21     }
22 }
```