MOV02 - Laboratorio 7

Juan Pablo Arias Mora

Febrero 2021

1 Introducción

Swift nos presenta el decorardor **State**, el propósito de este laboratorio es ejemplificar el concepto visto en clase. Y como este nos facilita el desarrollo en conjunto con **SwiftUI**.

Para referencias futuras el código fuente final del mismo esta disponible en:
https://github.com/pabloariasmora/MOV02-Cenfotec-Demo-Labs.git

1.1 Versiones

- MacOs 10.15.7 - Inglés
- Xcode 12.4
- Simulador iPhone 12 Pro Max

2 Instrucciones

"Algunos creen que enfocarse significa centrarse solamente en lo que ilusiona. Pero no es así, para enfocarse es necesario decir no a miles de ideas, por buenas que sean." Steve Jobs

2.1 Creación de aplicación base

Paso 1: Cree un nuevo proyecto en Xcode, utilizando el template App dentro de iOS, en la ventana de opciones para la configuración especifica de la nueva aplicación.

Product Name : swstate
Organization Identifier : com.cenfotec.mov02.07
Interface : SwiftUI
Life Cycle : SwiftUI App
Languge : Swift
Use Core Data: Sin Seleccionar
Include Test: Sin Seleccionar

Paso 2: Abrir el archivo **ContentView.swift**. (El código en el archivo debe ser similar al que se muestra a continuación)

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        Text("Hello, world!")
            .padding()
    }
}

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}
```

Paso 3: Dentro de la propiedad body, cambie el valor **"Hello, World!"** por el código que se muestra a continuación.En el código podemos observar como se relacionan los **Closures** con los **View**, para ello preste atención al area de **Action** y **Label** los cuales con parámetros del **Initializer** del botón y reciben **Closures**. El primero se ejecutará cuando se presione el botón. El segundo es para poder devolver una Vista dentro de él que define la apariencia de nuestro botón.

```
VStack {
      Text("Random Pokemon Generator")
      Text("---")
      Button(
        action: { print("Gotta catch 'em all") },
        label: { Text("Run") }
      )
    }
```

Paso 4: Compile y ejecute la aplicación. Que sucede cuando presiona el botón de **Run**?. A nivel de consola **Debug** se presenta un mensaje con el texto **"Gotta catch 'em all"**.

Paso 5: Agreguemos el siguiente código al segundo **Text** que nos presenta el valor de —. Esto nos permite un poco mejorar la interfaz de usuario.

```
.frame(
  width: UIScreen.main.bounds.width,
  height: 50
)
.background(Color.blue)
.foregroundColor(Color.white)
.padding(10)
```

Paso 6: Compilamos y ejecutamos el resultado debe ser similar a la figura 1. A nivel de código debe ser similar a 2.

Figure 1: View Principal

```
struct ContentView: View {
    var body: some View {
        VStack {
            Text("Random Pokemon Generator")
            Text("---")
              .frame(
                width: UIScreen.main.bounds.width,
                height: 50
              )
              .background(Color.blue)
              .foregroundColor(Color.white)
              .padding(10)
            Button(
              action: { print("gotta catch 'em all") },
              label: { Text("Run") }
            )
        }
    }
}

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}
```

Figure 2: Código 1

Paso 7: Vamos a definir una nueva variable a nivel del **ContentView** utilizando el siguiente código:

```
var pokemonName = "Charmander"
```

Paso 8: Vamos a definir reemplazar la definición del segundo **Text** utilizando esta nueva definición dependiente de la variable que definimos previamente.

```
Text(pokemonName)
```

Paso 9: Compilamos y ejecutamos el código, el resultado debe de ser un despliegue del valor inicial de la variable por parte del segundo **Text**.

Paso 10: Agreguemos la siguiente definición de la función switchPokemon como metodo del ContentView.

```
func switchPokemon() {
    pokemonName = "Pikachu"
}
```

Paso 11: A este punto un mensaje de error similar a la figura 3, en la linea 31.

Figure 3: Error

Paso 12: Esto se debe a que los **Struck** de forma predeterminada no pueden mutar sus propias propiedades. Aunque en este caso es un error que realmente no muestra lo que esta sucediendo. Para solucionar este error vamos a agregar el decorador **@State** en frente de la definición de la variable **pokemonName**. Y listo error eliminado.

Paso 13: Compilamos y ejecutamos el código. Qué paso? Una vez presionamos el botón de Run no hay ningún cambio aparente, pero si hemos solucionado el error ahora que sucede. Pues claro hemos realizado la definición de la variable y una función que cambia su valor pero realmente no hemos gestionado que el botón realice un llamado a la función. Justo después del print dentro del Closure en Action colocamos el siguiente código:

```
self.switchPokemon()
```

Paso 14: Compilamos y ejecutamos el código, como resultado ahora que presionamos el botón Run, podemos ver como el texto cambia a ser Pikachu. (ver 4), esto se debe a que cuando la variable cambia (debido al @State), por tanto invalida su visualización en la View y se actualiza nuevamente.

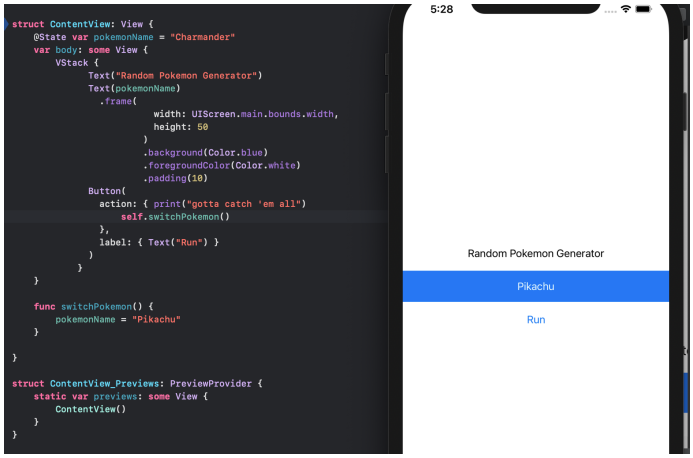

Figure 4: Cambio accionado desde el botón

Paso 15: Por ultimo lo que vamos a realizar es un selector de nombres de Pokemon de manera aleatoria basados en una lista, entonces vamos a actualizar la definición de la función switchPokemon con el siguiente código:

```
let list = ["Squirtle", "Bulbasaur", "Charmander", "Pikachu"]
pokemonName = list.randomElement() ?? ""
```

Paso 16: También no olvidemos actualizar el valor inicial de la variable pokemonName con el valor "—", para no mostrar un valor hasta que se presione por primera vez el botón. Como nota importante podría parecer que en algunos momento el valor no actualizará al presionar el botón esto se debe a que la propiedad **randomElement** podría retornar el mismo valor dada su semilla y la cantidad de opciones disponible.