CESAR HUMBERTO HERRERA VILLACORTA

1. Hablando exclusivamente del ámbito mobile, podemos mencionar a SwiftUI, Google Jetpack Compose y Flutter (aunque no nos olvidemos de React Native).
2. SwiftUI es un framework de interfaz declarativo creado por Apple. UIKit es la anterior tecnología (mas tradicional) utilizada para el desarrollo de aplicaciones iOS y macOS y que tenia una programación imperativa. Podemos decir que SwiftUI es la evolución de UIKit.
3. En términos de arquitectura son muy parecidos, ya que ambos representan el modelo de un objeto. Una clase, mas usado en el contexto de java, es un modelo que posee atributos y métodos. Una Struct es el equivalente a una clase, solo que en el contexto de Swift.
4. Para cambiar el tamaño de una imagen en SwiftUI, usamos el modificador ".resizable()". También podemos agregar parámetros como width y height.
5. Para hacer que una imagen se vea en forma circular en SwiftUI, puedes usar el modificador ".clipShape(Circle())".
6. Para realizar la navegación correctamente usamos NavigationStack (que será el contenedor) y NavigationLink (que serán los nodos de navegación).
7. @State en SwiftUI se usa para indicar que son propiedades cuyo cambio debe reflejarse en la vista. Esto permite que la vista se renderice nuevamente cuando el valor de la variable cambia.
8. En el curso usamos el patrón "Modelo-Vista VistaModelo" (Model-View ViewModel). MVVM separa la lógica de presentación de la vista y el modelo de datos, lo que permite escribir la interfaz y la lógica con poco acoplamiento.
9. En clase hemos usado el modificador ".photosPicker()" que permite guardar una foto de la galería en una variable de tipo "PhotosPickerItem".
10. Swift es un lenguaje de programación (como Kotlin o Java), mientras que SwiftUI es un framework diseñado para crear interfaces de usuario de manera declarativa (como Jetpack Compose).
11. ZStack se utiliza en SwiftUI para apilar vistas una encima de la otra en el eje Z (profundidad) para diseñar componentes superpuestos.
12. LazyVStack es utilizado cuando debemos cargar elementos de manera dinámica (a través de una llamada a una API por ejemplo), mientras que VStack se utiliza para una cantidad fija de elementos.
13. Spacer se utiliza para distribuir el espacio disponible de manera equitativa entre las vistas en un HStack o VStack. Ayuda a ajustar la distribución de espacio entre vistas.
14. Ambos son una especie de contenedores. TabView es una vista que contiene elementos TabItem. Cuando creamos una vista podemos incluirla dentro de la TabView y agregar el modificador .tabItem() para

convertirla en un TabItem. En otras palabras un TabView es una vista que contiene otras vistas (TabItems).

15. La carpeta utilizada para guardar los recursos (imágenes, iconos u otros archivos multimedia) es “Assets”.
16. Para permitir que el usuario ingrese datos usamos la vista TextField().
17. @Environment(\.dismiss) fue usado durante el curso para poder cambiar el botón “Back” predeterminado por uno customizado. Lo hemos asociado a un TapGesture para que al dar click, la vista se “cierre” y regrese a la vista anterior. Usando el código self.dismiss() la vista es eliminada de la pantalla.
18. Para que el usuario ingrese una contraseña usamos el elemento SecureField().
19.

```
VStack {
        HStack {
           Image(“imagen1”)
              .frame(width: 50, height: 50)
           Spacer()
           Image(“imagen2”)
              .frame(width: 50, height: 50)
        }
        Spacer()
     }
```

20.

```
TabView {
        Text(“Pestaña 1”)
           .tabItem {
              Image(systemName: "house")
           }
        Text("Pestaña 2”)
           .tabItem {
              Image(systemName: "calendar")
           }
        Text("Pestaña 3”)
           .tabItem {
              Image(systemName: "plus.square")
           }
     }
```