**Sesión 2-01 Clase del 16 de octubre**

En esta sesión vamos a ver cómo se trabaja con componentes **LazyRow** y **LazyColumn**. Son componentes que permiten mostrar listas grandes de componentes de forma eficiente.

1.- A partir de una lista con los nombres de 30 superheroes, vamos a mostrar esos nombres en componentes **Text** dentro de un contenedor **Column**.

val nombresDeSuperheroes = *listOf*(  
 "Spider-Man", "Iron Man", "Captain America", "Thor", "Hulk",  
 "Black Widow", "Doctor Strange", "Scarlet Witch", "Black Panther", "Ant-Man",  
 "Captain Marvel", "Hawkeye", "Vision", "War Machine", "Falcon",  
 "Winter Soldier", "Star-Lord", "Gamora", "Rocket Raccoon", "Groot",  
 "Drax", "Mantis", "Nebula", "Wasp", "Shang-Chi",  
 "Deadpool", "Wonder Woman", "Superman", "Batman", "Aquaman"  
)

Cada **Text** tiene un color de **background** de tono azulado, las esquinas algo redondeadas y ocupa todo el ancho de pantalla. El texto tiene un tamaño de 20 unidades.

@Composable  
fun TextSuperheroe(nombre: String) {  
 Text(  
 text = nombre,  
 fontSize = 20.*sp*,   
 color = Color.White,   
 modifier = Modifier  
 .*fillMaxWidth*()   
 .*background*(  
 color = *Color*(0xFF8899FF),   
 shape = *RoundedCornerShape*(8.*dp*)   
 )  
 .*padding*(16.*dp*)   
   
 )  
}

Antes de la lista pintamos un título “Lista de Superheroes” en tamaño de letra bastante grande.

2.- Al ejecutar veremos que solo se pueden ver los componentes que caben en la pantalla. La pantalla debería tener **Scroll**. Para poner scroll, bastaría con establecer en el modificador de la **Columna** que sea **scrollable**.

.*verticalScroll*(rememberScrollState())

3.- Si embargo y especialmente cuando una lista tiene muchos elementos y cada elemento contiene varios componentes, incluidas imágenes, el contenedor **Column** es ineficiente.

En su lugar se usa el **LazyColumn** (**Column** de carga perezosa). Este contenedor prepara sólo los elementos que el usuario está viendo. Cuando el usuario hace scroll va preparando los que necesita. Con muchos elementos, usando **Column** observaríamos que la lista tarda en verse. Con **LazyColumn** no apreciamos el retardo ya que inicialmente se cargan unos pocos elementos. Además, con cada recomposición de los elementos, **Column** vuelve a procesarlos todos.

Sustituiremos **Column** por **LazyColumn**. En **LazyColumn** no se tiene que especificar que es **Scrollable**, lo es ya por defecto. El código para implementar **LazyColumn** es:

LazyColumn(

verticalArrangement = Arrangement.spacedBy(8.*dp*),

modifier = Modifier  
 .*fillMaxWidth*()  
 .*padding*(16.*dp*) // Padding alrededor de toda la columna  
) **{** *items*(nombresDeSuperheroes)**{** name**->** TextSuperheroe(nombre = name)  
 **}  
}**

La función **ítems** itera a través de la lista y, usando una lambda, para cada elemento de la lista, pasa el contenido del elemento a la función para representar el **Text** del superheroe.

4.- Usaremos la función **item** dentro del **LazyColumn** para poner una fila de encabezado en la lista.

item **{** CabeceraLazyColumn() **}**

5.- Modificamos el **LazyColumn** para que, por cada superhéroe, se pinte a la derecha un **Checkbox** desmarcado. En cada elemento de LazyColumn se sustituirá el TextBox que tenemos actualmente por un Row que incluye el TextBox anterior y el nuevo **Checkbox**.

Row (verticalAlignment = Alignment.CenterVertically,  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*background*(  
 color = *Color*(0xFF8899FF),  
 shape = *RoundedCornerShape*(8.*dp*)  
 ))**{ }**

Al igual que ocurría con el componente de entrada **Textfield**, con el componente de entrada **Checkbox** tenemos que usar una variable de estado, en este caso de tipo **Boolean**.

// Estado del checkbox  
var isChecked by remember **{** *mutableStateOf*(false) **}**

El código base que usaremos para incluir el **Checkbox** es:

Checkbox(checked = isChecked,  
 onCheckedChange = **{**checked**->** isChecked=checked})

Comprobamos la funcionalidad.

6.- Ahora programamos en cada **Checkbox** cargado que al marcarlo o desmarcarlo escriba en pantalla un mensaje Toast que indique que el personaje correspondiente ha sido marcado o desmarcado.

Si quisiéramos programar funcionalidades como, por ejemplo, eliminar todos los que estén marcados, tendríamos que tener una variable de estado de tipo colección que tenga todos los elementos marcados o bien todos los existentes con su estado true o false. En otro momento lo usaremos.

7.- En un nuevo proyecto Sesión 2-01b vamos a probar el componente **LazyRow**.

Con un LazyRow implementaremos una “storie” de las imágenes supuestamente más recientes.

En el **LazyRow** se representarán en botones redondos 10 imágenes almacenadas en **res/drawable.** Las imágenes las tienes en la carpeta de recursos de Moodle.

Este es el código para implementar funcionalidad de un Botón redondo (usando un **Box** con una imagen)

@Composable  
fun BotonConImagen(imageResId: Int, onClick: () -> Unit) {  
 Box(  
 modifier = Modifier  
 .*size*(100.*dp*) // Tamaño del botón redondo  
 .*clip*(*CircleShape*) // Botón redondo  
 .*background*(Color.LightGray)

.*clickable* **{** onClick() **}** // Acción al hacer clic  
 ) **{** Image(  
 painter = painterResource(id = imageResId),  
 contentDescription = null,  
 modifier = Modifier.*fillMaxSize*(),  
 contentScale = ContentScale.Crop // Ajustar la imagen dentro del botón  
 )  
 **}**}

Prueba la funcionalidad, llamando a esa función con:

8.- Una vez que la función funciona, crea un **LazyRow** que carga las 10 imágenes previamente declaradas en un **List<Int>**

LazyRow(  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*padding*(10.*dp*),  
 horizontalArrangement = Arrangement.spacedBy(8.*dp*) // Espaciado entre botones  
) **{ }**

Recuerda que deberás usar la función ítems dentro del scope de **LazyRow** y programarla.

9.- Organizamos el diseño para que debajo de la fila que hemos creado, se haga un marco **Box** que ocupa todo el espacio restante de la pantalla y dentro del **Box** un componente **Image**.

10.- Programamos que cuando se clique un botón del storie, se muestre la imagen correspondiente en el marco.