Programación multimedia y dispositivos móviles

PMDM

2º DAM

IES Las Espeñetas
Practica:PR401-402
Profesor: Ginés
Alumno: Chuan Liang Lin Yang
6 de diciembre del 2023

ÍNDICE

Contents

Practica 401

Se pide desarrollar una aplicación gráfica que sea el cuaderno de un profesor. Para ello se deberá generar un menú que permita las siguientes acciones: • Poner notas • Sacar la nota más alta y la posición • Calcular la media de la clase quitando la nota más alta y más baja. • Borrar una nota en concreto • Borrar todas.

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Practica401Theme {
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    App()
                }
            }
        }
    }
}

class Alumno(var notas: DoubleArray = DoubleArray( size: 0)) {

}

@OptIn(ExperimentalMaterial3Api::class, ExperimentalComposeUiApi::class)
@Composable
fun App() {
    var alumno by remember { mutableStateOf(Alumno()) }
    var notasString by remember { mutableStateOf( value: "") }
    var media by remember { mutableStateOf( value: 0.0) }
    var mostrarNotasMasAltas by remember { mutableStateOf( value: false) }
    var mostrarMedia by remember { mutableStateOf( value: false) }
    var notasMasAltasText by remember { mutableStateOf( value: "") }
    var mostrarSeccionBorrarNotas by remember { mutableStateOf( value: false) }
    var mensajeError: String by remember { mutableStateOf( value: "") }
    var mensajeErrorExcepcion: String by remember { mutableStateOf( value: "") }
```

```
Column(
    modifier = Modifier
        .fillMaxSize()
        .padding(16.dp)
) { this: ColumnScope
    // Agregar notas
    TextField(
        value = notasString,
        onValueChange = { it: String
            notasString = it
        },
        label = { Text( text: "Introduce las notas separadas por comas") },
        keyboardOptions = KeyboardOptions.Default.copy(
            keyboardType = KeyboardType.Number,
            imeAction = ImeAction.Done
        ),
        keyboardActions = KeyboardActions(
            onDone = { this: KeyboardActionScope
                val notasArray =
                    notasString.split( ...delimiters: ",").map { it.trim().toDouble() }.toDoubleArray()
                alumno = Alumno(notasArray)
            }
        ),
        modifier = Modifier
            .fillMaxWidth()
            .padding(bottom = 8.dp)
    )

    Button(
        onClick = {
            val notasArray = notasString.split( ...delimiters: ",").map { it: String
                it.trim().toDoubleOrNull()
            }

            if (notasArray.all { it != null }) {
                // Todas las notas son números válidos
                alumno = Alumno(notasArray.requireNoNulls().toDoubleArray())

                // Vaciar el contenido del TextField después de enviar las notas
                notasString = ""
```

```
                // Reiniciar la visibilidad de las secciones
                mostrarNotasMasAltas = false
                mostrarMedia = false
                mensajeErrorExcepcion = ""
            } else {
                // Mostrar un mensaje de error si alguna nota no es un número válido
                mensajeErrorExcepcion = "Ingrese solo números válidos separados por comas."
            }
        },
        modifier = Modifier
            .fillMaxWidth()
            .padding(bottom = 8.dp)
    ) { this: RowScope
        Text( text: "Enviar notas")
    }
    // Mostrar el array de notas introducido
    if (alumno.notas.isNotEmpty()) {
        Text(
            text = "Array de notas: ${alumno.notas.joinToString( separator: ", ")}",
            modifier = Modifier.padding(bottom = 8.dp)
        )
    }
    if (mensajeErrorExcepcion.isNotEmpty()) {
        Text(
            text = mensajeErrorExcepcion,
            color = Color.Red,
            modifier = Modifier.padding(bottom = 8.dp)
        )
    }

    // Botón para sacar la nota más alta y su posición
    Button(
        onClick = {
            if (alumno.notas.isNotEmpty()) {
                var maxNota = alumno.notas[0]
                var posicionMax = 0

                for (i in 1 ≤ until  < alumno.notas.size) {
                    if (alumno.notas[i] > maxNota) {
                        maxNota = alumno.notas[i]
```

```
                        posicionMax = i
                    }
                }

                // Actualizar el estado con las notas más altas
                notasMasAltasText = "Notas más altas: $maxNota"

                // Mostrar la sección de notas más altas
                mostrarNotasMasAltas = true
            }
        },
        modifier = Modifier
            .fillMaxWidth()
            .padding(bottom = 8.dp)
    ) { this: RowScope
        Text( text: "Obtener nota más alta y posición")
    }


    // Botón para calcular la media sin la nota más alta y más baja
    Button(
        onClick = {
            if (alumno.notas.isNotEmpty()) {
                var maxNota = alumno.notas[0]
                var minNota = alumno.notas[0]
                var sumaNotas = 0.0
                var cantidadNotas = 0

                // Encontrar la nota más alta y más baja, y calcular la suma de las notas
                for (nota in alumno.notas) {
                    if (nota > maxNota) {
                        maxNota = nota
                    }
                    if (nota < minNota) {
                        minNota = nota
                    }
                    sumaNotas += nota
                    cantidadNotas++
                }
```

```
                // Restar la nota más alta y más baja de la suma
                sumaNotas -= maxNota
                sumaNotas -= minNota
                cantidadNotas -= 2 // Restar dos porque eliminamos dos notas (más alta y más baja)

                // Calcular la media sin la nota más alta y más baja
                val promedio = if (cantidadNotas > 0) {
                    sumaNotas / cantidadNotas.toDouble()
                } else {
                    // Manejar el caso cuando hay menos de dos notas
                    0.0
                }

                // Mostrar la sección de la media
                mostrarMedia = true
                media = promedio
            }
        },
        modifier = Modifier
            .fillMaxWidth()
            .padding(bottom = 8.dp)
    ) { this: RowScope
        Text( text: "Calcular media sin la nota más alta y más baja")
    }


    // Texto para mostrar las notas más altas
    if (mostrarNotasMasAltas) {
        Text(notasMasAltasText)
    }

    // Texto para mostrar la media
    if (mostrarMedia) {
        Text( text: "Media: $media")
    }

    // Botón para mostrar la sección de borrar notas
    Button(
        onClick = {
            mostrarSeccionBorrarNotas = true
```

```
        },
        modifier = Modifier
            .fillMaxWidth()
            .padding(bottom = 8.dp)
    ) { this: RowScope
        Text( text: "Mostrar sección de borrar notas")
    }

// Sección de borrar notas condicionalmente visible
    if (mostrarSeccionBorrarNotas) {
        // Campo de texto para introducir la posición de la nota a borrar
        var posicionBorrar by remember { mutableStateOf( value: "") }

        TextField(
            value = posicionBorrar,
            onValueChange = { it: String
                posicionBorrar = it
            },
            label = { Text( text: "Introduce la posición de la nota a borrar") },
            keyboardOptions = KeyboardOptions.Default.copy(
                keyboardType = KeyboardType.Number,
                imeAction = ImeAction.Done
            ),
            modifier = Modifier
                .fillMaxWidth()
                .padding(bottom = 8.dp)
        )
        // Botón para borrar la nota de la posición especificada
        Button(
            onClick = {
                // Al hacer clic en el botón, intenta borrar la nota en la posición especificada
                val posicion = posicionBorrar.toIntOrNull()
                if (posicion != null && posicion >= 0 && posicion < alumno.notas.size) {
                    alumno.notas = alumno.notas.filterIndexed { index, _ -> index != posicion }.toDoubleArray()
                    // Restablecer el campo de texto después de borrar
                    posicionBorrar = ""
                } else {
                    mensajeError = "Posición inválida"
                }
                posicionBorrar=""
```

```
            },
            modifier = Modifier
                .fillMaxWidth()
                .padding(bottom = 8.dp)
        ) { this: RowScope
            Text( text: "Borrar nota en la posición especificada")
        }
        if (mensajeError.isNotEmpty()) {
            Text(
                text = mensajeError,
                color = Color.Red,
                modifier = Modifier.padding(bottom = 8.dp)
            )
        }
        Button(
            onClick = {
                alumno = Alumno()
                mostrarMedia=false
                mostrarNotasMasAltas=false
            },
            modifier = Modifier
                .fillMaxWidth()
                .padding(bottom = 8.dp)
        ) { this: RowScope
            Text( text: "Borrar todas las notas")
        }
    }
}
}
```

Video del funcionamiento:

https://youtu.be/Jt2dF_-v8-s

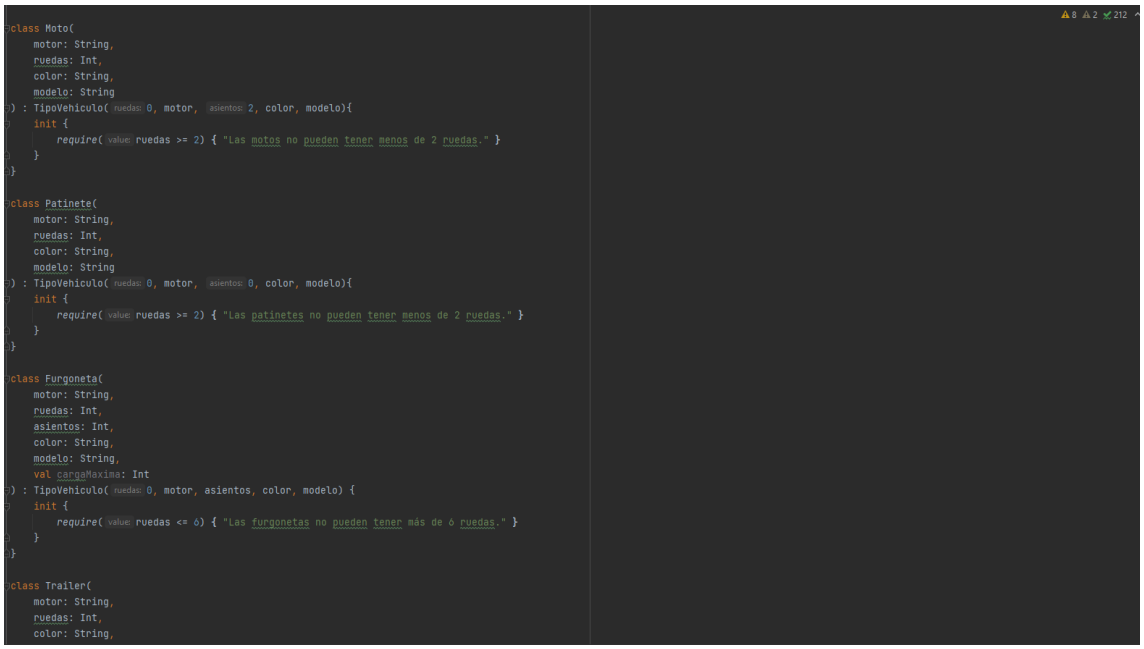
Practica 402

Se pide desarrollar una aplicación gráfica que sea el control de un concesionario que vende: coches, motos, patinetes, furgonetas y tráileres. Esta aplicación ha de permitir: - Crear el vehículo - Consultar el numero de los que se han creado de cada tipo. - Listado de los creados por nombre. Se ha de generar una clase llamada vehículo de la que heredarán el resto de las generadas. Cada vehículo ha de tener al menos: rueda, motor, número de asientos, color y modelo. Se han de tener las siguientes consideraciones: - En el caso de los patinetes se ha de controlar que no se ponga asiento. - En el de los tráileres que al menos tengan 6 ruedas, además de introducir la carga máxima. - En las furgonetas que tengan como máximo 6 ruedas y se necesita saber la carga máxima. - Las motos no pueden tener más de dos asientos

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            PR402Theme {
                // A surface container using the 'background' color from the theme
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    App()
                }
            }
        }
    }
}

// Clase base para representar un tipo genérico de vehículo
open class TipoVehiculo(
    val ruedas: Int,
    val motor: String,
    val asientos: Int,
    val color: String,
    val modelo: String
)

// Clases especificas que heredan de TipoVehiculo para representar diferentes tipos de vehiculos
class Coche(
    motor: String,
    asientos: Int,
    color: String,
    modelo: String
) : TipoVehiculo( ruedas: 4, motor, asientos, color, modelo) {
    init {
        require( value: ruedas >= 4) { "Las coches no pueden tener menos de 4 ruedas." }
    }
}
```

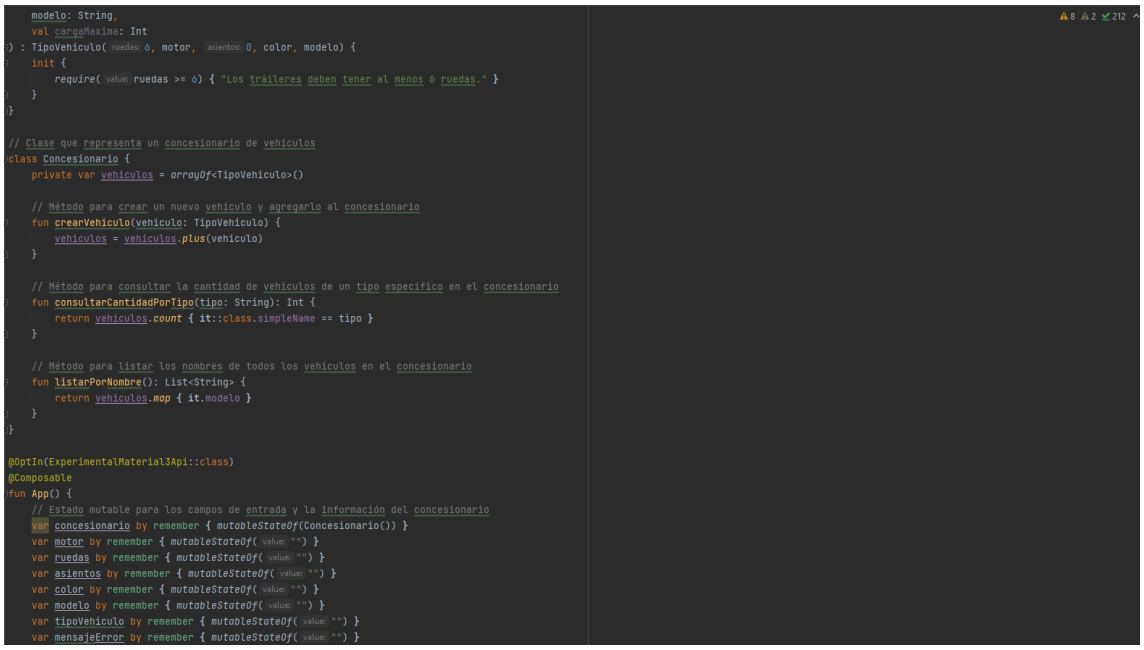


```
class Moto(
    motor: String,
    ruedas: Int,
    color: String,
    modelo: String
) : TipoVehiculo( ruedas: 0, motor,  asientos: 2, color, modelo){
    init {
        require( value: ruedas >= 2) { "Las motos no pueden tener menos de 2 ruedas." }
    }
}

class Patinete(
    motor: String,
    ruedas: Int,
    color: String,
    modelo: String
) : TipoVehiculo( ruedas: 0, motor,  asientos: 0, color, modelo){
    init {
        require( value: ruedas >= 2) { "Las patinetes no pueden tener menos de 2 ruedas." }
    }
}

class Furgoneta(
    motor: String,
    ruedas: Int,
    asientos: Int,
    color: String,
    modelo: String,
    val cargaMaxima: Int
) : TipoVehiculo( ruedas: 0, motor, asientos, color, modelo) {
    init {
        require( value: ruedas <= 6) { "Las furgonetas no pueden tener más de 6 ruedas." }
    }
}

class Trailer(
    motor: String,
    ruedas: Int,
    color: String,
```



```
    modelo: String,
    val cargaMaxima: Int
) : TipoVehiculo( ruedas: 6, motor,  asientos: 0, color, modelo) {
    init {
        require( value: ruedas >= 6) { "Los tráileres deben tener al menos 6 ruedas." }
    }
}

// Clase que representa un concesionario de vehículos
class Concesionario {
    private var vehiculos = arrayOf<TipoVehiculo>()

    // Método para crear un nuevo vehículo y agregarlo al concesionario
    fun crearVehiculo(vehiculo: TipoVehiculo) {
        vehiculos = vehiculos.plus(vehiculo)
    }

    // Método para consultar la cantidad de vehículos de un tipo específico en el concesionario
    fun consultarCantidadPorTipo(tipo: String): Int {
        return vehiculos.count { it::class.simpleName == tipo }
    }

    // Método para listar los nombres de todos los vehículos en el concesionario
    fun listarPorNombre(): List<String> {
        return vehiculos.map { it.modelo }
    }
}

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun App() {
    // Estado mutable para los campos de entrada y la información del concesionario
    var concesionario by remember { mutableStateOf(Concesionario()) }
    var motor by remember { mutableStateOf( value: "") }
    var ruedas by remember { mutableStateOf( value: "") }
    var asientos by remember { mutableStateOf( value: "") }
    var color by remember { mutableStateOf( value: "") }
    var modelo by remember { mutableStateOf( value: "") }
    var tipoVehiculo by remember { mutableStateOf( value: "") }
    var mensajeError by remember { mutableStateOf( value: "") }
```

```
    var listado by remember { mutableStateOf( value: "") }
    var cargaMaxima by remember { mutableStateOf( value: "") }

    // Columna que contiene varios campos de entrada y botones
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp)
    ) { this: ColumnScope
        // Campos de entrada para información del vehiculo
        TextField(
            value = motor,
            onValueChange = { motor = it },
            label = { Text( text: "Motor") },
            modifier = Modifier
                .fillMaxWidth()
                .padding(bottom = 8.dp)
        )

        TextField(
            value = ruedas,
            onValueChange = { ruedas = it },
            label = { Text( text: "Ruedas") },
            modifier = Modifier
                .fillMaxWidth()
                .padding(bottom = 8.dp)
        )

        TextField(
            value = asientos,
            onValueChange = { asientos = it },
            label = { Text( text: "Numero de asientos") },
            keyboardOptions = KeyboardOptions.Default.copy(
                keyboardType = KeyboardType.Number,
                imeAction = ImeAction.Done
            ),
            modifier = Modifier
                .fillMaxWidth()
                .padding(bottom = 8.dp)
```

```
        TextField(
            value = color,
            onValueChange = { color = it },
            label = { Text( text: "Color") },
            modifier = Modifier
                .fillMaxWidth()
                .padding(bottom = 8.dp)
        )

        TextField(
            value = modelo,
            onValueChange = { modelo = it },
            label = { Text( text: "Modelo") },
            modifier = Modifier
                .fillMaxWidth()
                .padding(bottom = 8.dp)
        )

        TextField(
            value = tipoVehiculo,
            onValueChange = { tipoVehiculo = it },
            label = { Text( text: "Tipo de Vehiculo") },
            modifier = Modifier
                .fillMaxWidth()
                .padding(bottom = 8.dp)
        )
        // Nuevo TextField para la carga máxima
        if (tipoVehiculo.lowercase() == "furgoneta" || tipoVehiculo.lowercase() == "trailer") {
            TextField(
                value = cargaMaxima,
                onValueChange = { cargaMaxima = it },
                label = { Text( text: "Carga Máxima") },
                keyboardOptions = KeyboardOptions.Default.copy(
                    keyboardType = KeyboardType.Number,
                    imeAction = ImeAction.Done
                ),
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(bottom = 8.dp)
            )
```

```
        // Botón para crear un nuevo vehículo en el concesionario
        Button(
            onClick = {
                try {
                    val ruedasInt = ruedas.toIntOrNull() ?: 0

                    // Verificación de carga máxima no vacía
                    if (tipoVehiculo.lowercase() == "furgoneta" || tipoVehiculo.lowercase() == "trailer") {
                        if (cargaMaxima.isBlank()) {
                            mensajeError = "La carga máxima no puede estar vacía."
                            return@Button
                        }
                    }

                    // Intentar convertir asientos a entero
                    val asientosInt = asientos.toInt()

                    mensajeError = ""

                    when (tipoVehiculo.lowercase()) {
                        "coche" -> concesionario.crearVehiculo(Coche(motor, asientosInt, color, modelo))
                        "moto" -> {
                            if (asientosInt <= 2) {
                                concesionario.crearVehiculo(Moto(motor, ruedasInt, color, modelo))
                            } else {
                                mensajeError = "Las motos no pueden tener más de 2 asientos."
                            }
                        }
                        "patinete" -> {
                            if (asientosInt == 0) {
                                concesionario.crearVehiculo(Patinete(motor, ruedasInt, color, modelo))
                            } else {
                                mensajeError = "Los patinetes no pueden tener asientos."
                            }
                        }
                        "furgoneta" -> {
                            val cargaMaximaInt = cargaMaxima.toIntOrNull() ?: throw NumberFormatException("La carga máxima no es un número válido.")
                            concesionario.crearVehiculo(Furgoneta(motor, ruedasInt, asientosInt, color, modelo, cargaMaximaInt))
                        }
```

```
                        "trailer" -> {
                            val cargaMaximaInt = cargaMaxima.toIntOrNull() ?: throw NumberFormatException("La carga máxima no es un número válido.")
                            concesionario.crearVehiculo(Trailer(motor, ruedasInt, color, modelo, cargaMaximaInt))
                        }
                        else -> mensajeError = "Tipo de vehiculo no reconocido"
                    }
                    motor=""
                    ruedas=""
                    asientos=""
                    color=""
                    modelo=""
                    tipoVehiculo=""
                } catch (e: NumberFormatException) {
                    mensajeError = "Error no puedes introducir algo que no sea numero: ${e.message}"
                    e.printStackTrace()
                } catch (e: Exception) {
                    mensajeError = "Error al crear el vehiculo: ${e.message}"
                    e.printStackTrace()
                }
            },
            modifier = Modifier
                .fillMaxWidth()
                .padding(bottom = 8.dp)
        ) { this: RowScope
            Text( text: "Crear Vehiculo")
        }

        // Campo para mostrar mensajes de error o información del concesionario
        Text(
            text = if (mensajeError.isNotEmpty()) mensajeError else "",
            modifier = Modifier.padding(bottom = 8.dp)
        )

        // Botón para consultar la cantidad de vehiculos por tipo y listar por nombre
        Button(
            onClick = {
                val coches = concesionario.consultarCantidadPorTipo( tipo: "Coche")
                val motos = concesionario.consultarCantidadPorTipo( tipo: "Moto")
                val patinetes = concesionario.consultarCantidadPorTipo( tipo: "Patinete")
                val furgonetas = concesionario.consultarCantidadPorTipo( tipo: "Furgoneta")
```

```
            val trailers = concesionario.consultarCantidadPorTipo( tipo: "Trailer")

            // Display the information in the UI
            listado = buildString { this: StringBuilder
                append("Cantidad de coches: $coches\n")
                append("Cantidad de motos: $motos\n")
                append("Cantidad de patinetes: $patinetes\n")
                append("Cantidad de furgonetas: $furgonetas\n")
                append("Cantidad de trailers: $trailers\n")
            }

            val vehiculosPorNombre = concesionario.listarPorNombre()
            listado += "Listado de vehiculos por nombre: $vehiculosPorNombre"
        },
        modifier = Modifier
            .fillMaxWidth()
            .padding(bottom = 8.dp)
    ) { this: RowScope
        Text( text: "Consultar y Listar")
    }

    // Campo para mostrar información de la consulta y listado
    Text(
        text = if (listado.isNotEmpty()) listado else "",
        modifier = Modifier.padding(bottom = 8.dp)
    )
  }
}
```

Video del funcionamiento:

https://youtu.be/jlWaMW3SrAw