UNIVERSIDAD TECNOLÓGICA DE LEÓN

INGENIERIA EN DESARROLLO Y GESTIÓN DE SOFTWARE

DESARROLLO PARA DISPOSITIBOS INTELIGENTES

Examen 2do Parcial

presenta:

Avila Juárez Kevin Francisco

IDGS903

Fecha: 03/07/2025

Zodiaco Chino

```
package com.example.zoodiacochino

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.activity.enableEdgeToEdge
import androidx.compose.runtime.Composable
import androidx.lifecycle.ViewModel
import androidx.lifecycle.ViewModelProvider
import androidx.lifecycle.viewmodel.compose.viewModel
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.compose.rememberNavController
import androidx.compose.ui.platform.LocalContext
import com.example.zoodiacochino.ui.theme.ZoodiacochinoTheme

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            ZoodiacochinoTheme {
                ZodiacoApp()
            }
        }
    }
}

@Composable
fun ZodiacoApp() {
    val navController = rememberNavController()
    val context = LocalContext.current

    val viewModel: ZodiacoViewModel = viewModel(
        factory = object : ViewModelProvider.Factory {
            @Suppress("UNCHECKED_CAST")
            override fun <T : ViewModel> create(modelClass: Class<T>): T {
                if (modelClass.isAssignableFrom(ZodiacoViewModel::class.java)) {
                    return ZodiacoViewModel(context) as T
                }
                throw IllegalArgumentException("Unknown ViewModel class")
            }
        }
    )
```

```
    NavHost(
        navController = navController,
        startDestination = "formulario"
    ) {
        composable("formulario") {
            FormularioScreen(
                onNavigateToExamen = {
                    viewModel.iniciarExamen()
                    navController.navigate("examen")
                },
                viewModel = viewModel
            )
        }

        composable("examen") {
            ExamenScreen(
                viewModel = viewModel,
                onFinishExam = {
                    navController.navigate("resultados")
                }
            )
        }

        composable("resultados") {
            ResultadosScreen(
                viewModel = viewModel,
                onNavigateToStart = {
                    navController.navigate("formulario") {
                        popUpTo("formulario") { inclusive = true }
                    }
                }
            )
        }
    }
}
```

```
package com.example.zoodiacochino

import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material3.Button
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.RadioButton
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.lifecycle.ViewModel
import androidx.lifecycle.ViewModelProvider
import androidx.lifecycle.viewmodel.compose.viewModel

@Composable
fun FormularioScreen(
    onNavigateToExamen: () -> Unit,
    viewModel: ZodiacoViewModel
) {
    val persona = viewModel.persona

    var nombre by remember { mutableStateOf(persona.nombre) }
    var apePaterno by remember { mutableStateOf(persona.apePaterno) }
    var apeMaterno by remember { mutableStateOf(persona.apeMaterno) }
    var dia by remember { mutableStateOf(persona.dia) }
    var mes by remember { mutableStateOf(persona.mes) }
    var anio by remember { mutableStateOf(persona.anio) }
    var sexo by remember { mutableStateOf(persona.sexo) }
```

```
Column(
    modifier = Modifier
        .fillMaxSize()
        .padding(16.dp),
    horizontalAlignment = Alignment.CenterHorizontally,
    verticalArrangement = Arrangement.Top
) {
    Text(
        text = "Datos Personales",
        fontSize = 24.sp,
        fontWeight = FontWeight.Bold,
        modifier = Modifier.padding(top = 26.dp, bottom = 24.dp)
    )

    val textFieldModifier = Modifier
        .weight(1f)
        .padding(vertical = 4.dp)

    val textFieldStyle = androidx.compose.ui.text.TextStyle(fontSize = 14.sp)

    Row(verticalAlignment = Alignment.CenterVertically) {
        Text(text = "Nombre: ", modifier = Modifier.padding(end = 8.dp))
        OutlinedTextField(
            value = nombre,
            onValueChange = { nombre = it },
            label = { Text("Nombre") },
            textStyle = textFieldStyle,
            modifier = textFieldModifier
        )
    }

    Row(verticalAlignment = Alignment.CenterVertically) {
        Text(text = "Apellido paterno: ", modifier = Modifier.padding(end = 8.dp))
        OutlinedTextField(
            value = apePaterno,
            onValueChange = { apePaterno = it },
            label = { Text("Apellido paterno") },
            textStyle = textFieldStyle,
            modifier = textFieldModifier
        )
    }

    Row(verticalAlignment = Alignment.CenterVertically) {
        Text(text = "Apellido materno: ", modifier = Modifier.padding(end = 8.dp))
        OutlinedTextField(
            value = apeMaterno,
```

```
            onValueChange = { apeMaterno = it },
            label = { Text("Apellido materno") },
            textStyle = textFieldStyle,
            modifier = textFieldModifier
        )
    }

    Text(
        text = "Fecha de nacimiento",
        fontWeight = FontWeight.SemiBold,
        modifier = Modifier.padding(top = 24.dp)
    )

    Row(
        horizontalArrangement = Arrangement.SpaceEvenly,
        modifier = Modifier
            .fillMaxWidth()
            .padding(vertical = 8.dp)
    ) {
        listOf(
            Triple("Día", dia, { it: String -> dia = it }),
            Triple("Mes", mes, { it: String -> mes = it }),
            Triple("Año", anio, { it: String -> anio = it })
        ).forEach { (label, value, onChange) ->
            Column(horizontalAlignment = Alignment.CenterHorizontally) {
                Text(text = label, textAlign = TextAlign.Center)
                OutlinedTextField(
                    value = value,
                    onValueChange = { if (it.all { c -> c.isDigit() }) onChange(it) },
                    singleLine = true,
                    modifier = Modifier.width(80.dp),
                    keyboardOptions = KeyboardOptions(keyboardType =
KeyboardType.Number),
                    textStyle = textFieldStyle
                )
            }
        }
    }

    Text(
        text = "Sexo",
        fontWeight = FontWeight.SemiBold,
        modifier = Modifier.padding(top = 16.dp)
    )

    Row(verticalAlignment = Alignment.CenterVertically) {
        RadioButton(
```

```
            selected = sexo == "Masculino",
            onClick = { sexo = "Masculino" }
        )
        Text(text = "Masculino")

        Spacer(modifier = Modifier.width(16.dp))

        RadioButton(
            selected = sexo == "Femenino",
            onClick = { sexo = "Femenino" }
        )
        Text(text = "Femenino")
    }

    Row(
        horizontalArrangement = Arrangement.SpaceEvenly,
        modifier = Modifier
            .fillMaxWidth()
            .padding(top = 24.dp)
    ) {
        Button(onClick = {
            nombre = ""
            apePaterno = ""
            apeMaterno = ""
            dia = ""
            mes = ""
            anio = ""
            sexo = ""
            viewModel.limpiarFormulario()
        }) {
            Text("Limpiar")
        }

        Button(onClick = {
            viewModel.actualizarPersona(
                ZodiacoViewModel.PersonaFormulario(
                    nombre = nombre,
                    apePaterno = apePaterno,
                    apeMaterno = apeMaterno,
                    dia = dia,
                    mes = mes,
                    anio = anio,
                    sexo = sexo
                )
            )
            anio.toIntOrNull()?.let {
                viewModel.calcularYGuardarSigno(it)
```

```
                onNavigateToExamen()
            }
        }) {
            Text("Siguiente")
        }
    }
}
```

```
package com.example.zoodiacochino

import androidx.compose.foundation.layout.*
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.verticalScroll
import androidx.compose.material3.Button
import androidx.compose.material3.Card
import androidx.compose.material3.CardDefaults
import androidx.compose.material3.RadioButton
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.lifecycle.viewmodel.compose.viewModel

@Composable
fun ExamenScreen(viewModel: ZodiacoViewModel, onFinishExam: () -> Unit) {
    val preguntasExamen = viewModel.preguntasExamen
    val respuestasUsuario = viewModel.respuestasUsuario
    val scrollState = rememberScrollState()

    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp)
    ) {
        Text(
            text = "Examen de Zodiaco Chino",
            fontSize = 24.sp,
            fontWeight = FontWeight.Bold,
            modifier = Modifier.padding(bottom = 16.dp)
        )

        Text(
            text = "Responde todas las preguntas",
            fontSize = 16.sp,
            modifier = Modifier.padding(bottom = 16.dp)
        )

        Column(
            modifier = Modifier
                .weight(1f)
                .verticalScroll(scrollState)
        ) {
```

```
        preguntasExamen.forEachIndexed { preguntaIndex, pregunta ->
            Card(
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(vertical = 8.dp),
                elevation = CardDefaults.cardElevation(defaultElevation = 4.dp)
            ) {
                Column(
                    modifier = Modifier.padding(16.dp)
                ) {
                    Text(
                        text = "${preguntaIndex + 1}. ${pregunta.enunciado}",
                        fontSize = 16.sp,
                        fontWeight = FontWeight.Medium,
                        modifier = Modifier.padding(bottom = 12.dp)
                    )

                    pregunta.opciones.forEachIndexed { opcionIndex, opcion ->
                        Row(
                            verticalAlignment = Alignment.CenterVertically,
                            modifier = Modifier
                                .fillMaxWidth()
                                .padding(vertical = 2.dp)
                        ) {
                            RadioButton(
                                selected = respuestasUsuario.getOrNull(preguntaIndex) ==
opcionIndex,
                                onClick = {
                                    val nuevasRespuestas =
respuestasUsuario.toMutableList()
                                    nuevasRespuestas[preguntaIndex] = opcionIndex
                                    viewModel.actualizarRespuestas(nuevasRespuestas)
                                }
                            )
                            Text(
                                text = "${('a' + opcionIndex).uppercase()}) $opcion",
                                modifier = Modifier.padding(start = 8.dp),
                                fontSize = 14.sp
                            )
                        }
                    }
                }
            }

            Spacer(modifier = Modifier.height(16.dp))
        }
```

```
        Button(
            onClick = {
                viewModel.terminarExamen()
                onFinishExam()
            },
            modifier = Modifier
                .fillMaxWidth()
                .padding(top = 16.dp),
            enabled = respuestasUsuario.all { it != -1 }
        ) {
            Text("Terminar Examen")
        }
    }
}
```

```
package com.example.zoodiacochino

import java.time.LocalDate

data class Persona(
    val nombre: String = "",
    val apePaterno: String = "",
    val apeMaterno: String = "",
    val fechaNacimiento: LocalDate,
    val sexo: String = ""
)
```

```
package com.example.zoodiacochino

data class Pregunta(
    val enunciado: String,
    val opciones: List<String>,
    val respuestaCorrecta: Int
)

object BancoPreguntas {
    val preguntas = listOf(
        Pregunta(
            "¿Cuál es la suma de 2 + 2?",
            listOf("8", "6", "4", "3"),
            2
        ),
        Pregunta(
            "¿Cuál es la capital de México?",
            listOf("Berlín", "CDMX", "Guadalajara", "Tirana"),
            1
        ),
        Pregunta(
            "¿Cuál es el planeta más grande del sistema solar?",
            listOf("Sol", "Saturno", "Júpiter", "Tierra"),
            2
        ),
        Pregunta(
            "¿Cuál es el resultado de 7 - 3?",
            listOf("2", "4", "12", "15"),
            1
        ),
        Pregunta(
            "¿Qué dorsal lleva Cristiano Ronaldo?",
            listOf("1", "7", "25", "8"),
            1
        ),
        Pregunta(
            "¿Cuál es la capital de Albania?",
            listOf("Guadalajara", "Madrid", "Tirana", "3"),
            2
        ),
        Pregunta(
            "¿Cuál es la suma de 5 + 3?",
            listOf("7", "8", "9", "10"),
            1
        ),
        Pregunta(
            "¿Qué dorsal usa Messi en la Selección Argentina?",
```

```
            listOf("10", "9", "19", "30"),
            0
        ),
        Pregunta(
            "¿Cuál es el resultado de 10 - 4?",
            listOf("5", "6", "7", "8"),
            1
        ),
        Pregunta(
            "¿Cuántos lados tiene un triángulo equilátero?",
            listOf("2", "3", "18", "5"),
            1
        )
    )
}
```

```
package com.example.zoodiacochino

import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

@Composable
fun ResultadosScreen(
    viewModel: ZodiacoViewModel,
    onNavigateToStart: () -> Unit
) {
    val persona = viewModel.persona
    val edad = viewModel.calcularEdad()
    val signo = viewModel.leerSignoGuardado() ?: "Desconocido"
    val calificacion = viewModel.calificacion

    val iconosSignos = mapOf(
        "Rata" to "🐭",
        "Buey" to "🐮",
        "Tigre" to "🐯",
        "Conejo" to "🐰",
        "Dragón" to "🐲",
        "Serpiente" to "🐍",
        "Caballo" to "🐴",
        "Cabra" to "🐐",
        "Mono" to "🐵",
        "Gallo" to "🐓",
        "Perro" to "🐶",
        "Cerdo" to "🐷"
    )

    Column(
        modifier = Modifier
```

```
            .fillMaxSize()
            .padding(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        Card(
            modifier = Modifier
                .fillMaxWidth()
                .padding(8.dp),
            shape = RoundedCornerShape(16.dp),
            elevation = CardDefaults.cardElevation(defaultElevation = 8.dp)
        ) {
            Column(
                modifier = Modifier.padding(16.dp),
                horizontalAlignment = Alignment.CenterHorizontally
            ) {
                Text(
                    text = "Hola ${persona.nombre} ${persona.apePaterno},
${persona.apeMaterno}",
                    fontSize = 20.sp,
                    fontWeight = FontWeight.Bold,
                    textAlign = TextAlign.Center
                )

                Spacer(modifier = Modifier.height(8.dp))

                Text(
                    text = "Tienes $edad años y tu signo zodiacal",
                    fontSize = 16.sp,
                    textAlign = TextAlign.Center
                )

                Spacer(modifier = Modifier.height(16.dp))

                iconosSignos[signo]?.let { icono ->
                    Box(
                        modifier = Modifier
                            .size(120.dp)
                            .background(
                                Color(0xFFF5F5F5),
                                CircleShape
                            ),
                        contentAlignment = Alignment.Center
                    ) {
                        Text(
                            text = icono,
                            fontSize = 60.sp
```

```
                )
            }
        }

        Spacer(modifier = Modifier.height(16.dp))

        Text(
            text = "Es $signo",
            fontSize = 18.sp,
            fontWeight = FontWeight.Bold
        )

        Spacer(modifier = Modifier.height(16.dp))

        Text(
            text = "Calificación $calificacion",
            fontSize = 24.sp,
            fontWeight = FontWeight.Bold,
            color = when {
                calificacion >= 8 -> Color(0xFF4CAF50)
                calificacion >= 6 -> Color(0xFFFF9800)
                else -> Color(0xFFF44336)
            }
        )
    }
}

Spacer(modifier = Modifier.height(24.dp))

Button(
    onClick = {
        viewModel.limpiarFormulario()
        onNavigateToStart()
    }
) {
    Text("Nuevo Examen")
}
}
}
```

```
package com.example.zoodiacochino

import android.content.Context
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.setValue
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import java.io.File

class ZodiacoViewModel(private val context: Context) : ViewModel() {

    private val fileName = "signo.txt"
    private val resultadosFileName = "resultados_examen.txt"

    var persona by mutableStateOf(PersonaFormulario())
        private set

    var preguntasExamen by mutableStateOf<List<Pregunta>>(emptyList())
        private set
    var respuestasUsuario by mutableStateOf<List<Int>>(emptyList())
        private set
    var preguntaActual by mutableStateOf(0)
        private set
    var examenTerminado by mutableStateOf(false)
        private set
    var calificacion by mutableStateOf(0)
        private set

    data class PersonaFormulario(
        val nombre: String = "",
        val apePaterno: String = "",
        val apeMaterno: String = "",
        val dia: String = "",
        val mes: String = "",
        val anio: String = "",
        val sexo: String = ""
    )

    fun actualizarPersona(nuevaPersona: PersonaFormulario) {
        persona = nuevaPersona
    }

    fun calcularEdad(): Int {
        val anioNacimiento = persona.anio.toIntOrNull() ?: 2000
```

```
    return 2024 - anioNacimiento
}

fun calcularYGuardarSigno(anio: Int) {
    val signosChinos = listOf(
        "Mono", "Gallo", "Perro", "Cerdo", "Rata", "Buey",
        "Tigre", "Conejo", "Dragón", "Serpiente", "Caballo", "Cabra"
    )
    val signo = signosChinos[anio % 12]

    viewModelScope.launch(Dispatchers.IO) {
        try {
            val file = File(context.filesDir, fileName)
            file.writeText(signo)
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }
}

fun leerSignoGuardado(): String? {
    return try {
        val file = File(context.filesDir, fileName)
        if (file.exists()) file.readText() else null
    } catch (e: Exception) {
        e.printStackTrace()
        null
    }
}

fun iniciarExamen() {
    preguntasExamen = BancoPreguntas.preguntas
    respuestasUsuario = List(10) { -1 }
    preguntaActual = 0
    examenTerminado = false
    calificacion = 0
}

fun responderPregunta(respuesta: Int) {
    if (preguntaActual < respuestasUsuario.size) {
        respuestasUsuario = respuestasUsuario.toMutableList().apply {
            this[preguntaActual] = respuesta
        }
    }
}

fun actualizarRespuestas(nuevasRespuestas: List<Int>) {
```

```
        respuestasUsuario = nuevasRespuestas
    }

    fun terminarExamen() {
        var respuestasCorrectas = 0
        for (i in preguntasExamen.indices) {
            if (respuestasUsuario[i] == preguntasExamen[i].respuestaCorrecta) {
                respuestasCorrectas++
            }
        }
        calificacion = (respuestasCorrectas * 10) / preguntasExamen.size
        examenTerminado = true

        guardarResultados(respuestasCorrectas, preguntasExamen.size)
    }

    private fun guardarResultados(correctas: Int, total: Int) {
        viewModelScope.launch(Dispatchers.IO) {
            try {
                val file = File(context.filesDir, resultadosFileName)
                val resultado = "${persona.nombre} ${persona.apePaterno}:
$correctas/$total - Calificación: $calificacion"
                file.writeText(resultado)
            } catch (e: Exception) {
                e.printStackTrace()
            }
        }
    }

    fun limpiarFormulario() {
        persona = PersonaFormulario()
    }
}
```