

TESCHA

INGENIERÍA EN SISTEMAS COMPUTACIONALES

MATERIA

Programación de Aplicaciones para Dispositivos

PROFESOR

Iván Azamar Palma

ALUMNO

Yáñez Moreno Alexis Gabriel

GRUPO:

4952

Tabla de contenido

MARCO TEÓRICO	4
Android Studio	4
SDK (Software Development Kit)	4
Kotlin	4
XML (Extensible Markup Language).....	5
Widgets.....	5
val	5
fun.....	5
Activity	6
onCreate.....	6
this	6
View.....	6
override fun	7
findViewById	7
Layout	7
EJERCICIOS KOTLIN.....	8
Ejercicio 01	8
Ejercicio 02	8
Ejercicio 03	9
Ejercicio 04	10
Ejercicio 05	11
Ejercicio 06	12
Ejercicio 07	14
Ejercicio 08	16
Ejercicio 09	17
Ejercicio 10	18
PROGRAMACIÓN ORIENTADA A OBJETOS EN KOTLIN.....	19
Data Class.....	20
Class Alumno Mejorado.....	22

Principal.....	22
APP Herencia	23
Herencia	24
IMC.....	25
Introducción	25
Desarrollo.....	26
Diseño en Android Studio	28
Plantilla De Diseño De Android Studio.	30
Código De La Mainactivity.....	30
Pantalla De Historial.....	31
Diseño Visual De La Pantalla Historial.....	32
Código De La Pantalla De Historial.	33
Funcionamiento De La Aplicación.....	33
Ingreso De Datos De La Primera Persona.	34
IMC Calculado.....	34
Verificación Del Funcionamiento El Historial.....	35
CONCLUSION	35

MARCO TEÓRICO

Android Studio

Android Studio es el entorno de desarrollo integrado (IDE) oficial para crear aplicaciones Android. Fue lanzado por Google en 2013 como reemplazo de Eclipse con el plugin ADT, con el fin de ofrecer una plataforma más completa y enfocada en este sistema operativo.

Entre sus principales características destacan el editor de código avanzado con autocompletado, las herramientas de diseño visual, el emulador de dispositivos y la integración con Gradle para gestionar proyectos. Además, soporta lenguajes como Java y Kotlin, lo que facilita el desarrollo de aplicaciones modernas.

Hoy en día, Android Studio es la herramienta más utilizada para programar en Android, ya que cuenta con soporte oficial, actualizaciones constantes y un entorno flexible que se adapta tanto a principiantes como a desarrolladores profesionales.

SDK (Software Development Kit)

Un SDK es un conjunto de herramientas de desarrollo que permite crear aplicaciones para una plataforma determinada. En el caso de Android, el Android SDK incluye librerías, compiladores, emuladores y documentación que facilitan la programación de aplicaciones móviles.

Fue presentado en 2008 junto con la primera versión del sistema operativo Android, y desde entonces se ha convertido en un recurso fundamental para la comunidad de desarrolladores. Actualmente, el SDK es indispensable en el desarrollo móvil, ya que ofrece las APIs necesarias para interactuar con las funciones del dispositivo, como la cámara, el GPS o los sensores, además de garantizar la compatibilidad de las aplicaciones con las diferentes versiones de Android.

Kotlin

Kotlin es un lenguaje de programación moderno creado por JetBrains y presentado en 2011. Destaca por ser conciso, seguro y totalmente interoperable con Java, lo que facilita su uso en proyectos existentes. En 2017, Google lo reconoció como lenguaje oficial para Android, impulsando su adopción en la industria.

Hoy en día, Kotlin es muy utilizado en el desarrollo móvil gracias a su sintaxis clara, su seguridad frente a valores nulos y la reducción de código repetitivo. Aunque es el preferido para aplicaciones Android, también se emplea en desarrollo web, de escritorio y multiplataforma.

XML (Extensible Markup Language)

XML (eXtensible Markup Language) es un lenguaje de marcado diseñado para almacenar, transportar y organizar datos de manera estructurada y legible tanto por humanos como por máquinas. Fue creado como una evolución más simple y flexible que SGML, y se ha convertido en un estándar para el intercambio de información. En Android, XML se utiliza principalmente para definir interfaces de usuario, estructuras de datos, recursos y configuraciones de la aplicación. Su principal ventaja es que permite separar el diseño visual de la lógica de programación escrita en Kotlin o Java, lo que facilita la mantenibilidad del código y el trabajo colaborativo entre diseñadores y programadores.

Widgets

Los widgets son componentes gráficos que permiten la interacción del usuario con la aplicación. Entre ellos se incluyen botones, listas, cuadros de texto, imágenes y menús desplegables. Introducidos desde las primeras versiones de Android, los widgets se diseñaron para crear interfaces modulares, reutilizables y consistentes con las guías de diseño de la plataforma. Hoy en día, son esenciales para construir aplicaciones intuitivas, funcionales y visualmente atractivas, ya que permiten combinar diferentes elementos de manera coherente y mejorar la experiencia del usuario.

val

En Kotlin, `val` es una palabra clave que se utiliza para declarar variables de solo lectura, es decir, valores que no pueden ser modificados después de ser inicializados. Esta característica fomenta la inmutabilidad, un principio importante para reducir errores y comportamientos inesperados en los programas. El uso de `val` ayuda a escribir código más seguro y confiable, promueve buenas prácticas de programación y favorece un estilo más claro y funcional, donde cada dato tiene un propósito definido y no puede cambiar accidentalmente.

fun

`fun` es la palabra clave utilizada en Kotlin para declarar funciones, que son bloques de código que realizan tareas específicas y pueden ser reutilizados en diferentes partes del programa. Las funciones permiten organizar el código en módulos claros, reduciendo la duplicación y mejorando la legibilidad. Además, `fun` facilita la

programación orientada a objetos y funcional, permitiendo que las aplicaciones sean más escalables y fáciles de mantener. La simplicidad de esta sintaxis diferencia a Kotlin de lenguajes más verbosos como Java, haciendo que el desarrollo sea más ágil y eficiente.

Activity

Una Activity es un componente fundamental en Android que representa una pantalla con la que el usuario puede interactuar. Cada Activity gestiona su propia interfaz, eventos y lógica de interacción, sirviendo como la unidad básica de navegación dentro de una aplicación. Introducidas desde la primera versión de Android, las Activities permiten organizar el flujo de la aplicación, controlar la interacción del usuario y administrar recursos como datos, vistas y servicios. Son la base del desarrollo móvil en Android, ya que cada pantalla, menú o función de la app suele estar ligada a una Activity específica.

onCreate

onCreate es un método del ciclo de vida de una Activity que se ejecuta al crear la actividad por primera vez. Es el lugar donde se inicializan los componentes principales de la pantalla, como la interfaz de usuario, variables, adaptadores y recursos necesarios para que la aplicación funcione correctamente. Este método es esencial, ya que garantiza que la Activity esté completamente configurada antes de que el usuario interactúe con ella. Entender onCreate es clave para controlar el ciclo de vida de una aplicación y optimizar el rendimiento y la experiencia del usuario.

this

En Kotlin, this es una referencia al objeto actual de la clase en la que se encuentra. Permite acceder a propiedades, métodos y el contexto de la instancia activa. En Android, this se utiliza frecuentemente dentro de Activities y Fragmentos para indicar el contexto actual, lo que es necesario para operaciones como inicializar vistas, crear intents, mostrar diálogos o acceder a recursos del sistema. Su correcto uso evita confusiones entre variables locales y propiedades de clase, asegurando un código más claro y organizado.

View

Una View es la unidad básica de la interfaz gráfica en Android, representando cualquier elemento visible en la pantalla, como botones, textos, imágenes o

contenedores. Todos los widgets y componentes interactivos se derivan de la clase View. Las Views permiten construir interfaces dinámicas, responsivas y accesibles, ya que se pueden combinar, personalizar y adaptar a distintos tamaños de pantalla y resoluciones. Comprender la clase View es esencial para desarrollar aplicaciones Android que sean eficientes y atractivas visualmente.

override fun

override fun se utiliza en Kotlin para sobrescribir métodos que han sido definidos en una clase padre o en una interfaz. La palabra override indica que la función redefine un comportamiento existente, mientras que fun declara la función como tal. Esta característica es fundamental en la programación orientada a objetos, permitiendo adaptar o mejorar la funcionalidad de métodos heredados. En Android, se utiliza comúnmente para modificar el comportamiento de métodos del ciclo de vida de una Activity, como onCreate o onStart, o para implementar funciones de interfaces, lo que mejora la extensibilidad y flexibilidad del código.

findViewById

findViewById es un método que permite obtener referencias a los componentes de la interfaz definidos en archivos XML mediante su identificador (id). Introducido en las primeras versiones de Android, conecta el diseño visual con la lógica de programación en Kotlin o Java. Aunque hoy existen métodos más modernos como View Binding y Data Binding, findViewById sigue siendo importante para comprender la estructura básica del desarrollo de interfaces, enseñar cómo interactúan el diseño y el código, y mantener compatibilidad con proyectos antiguos.

Layout

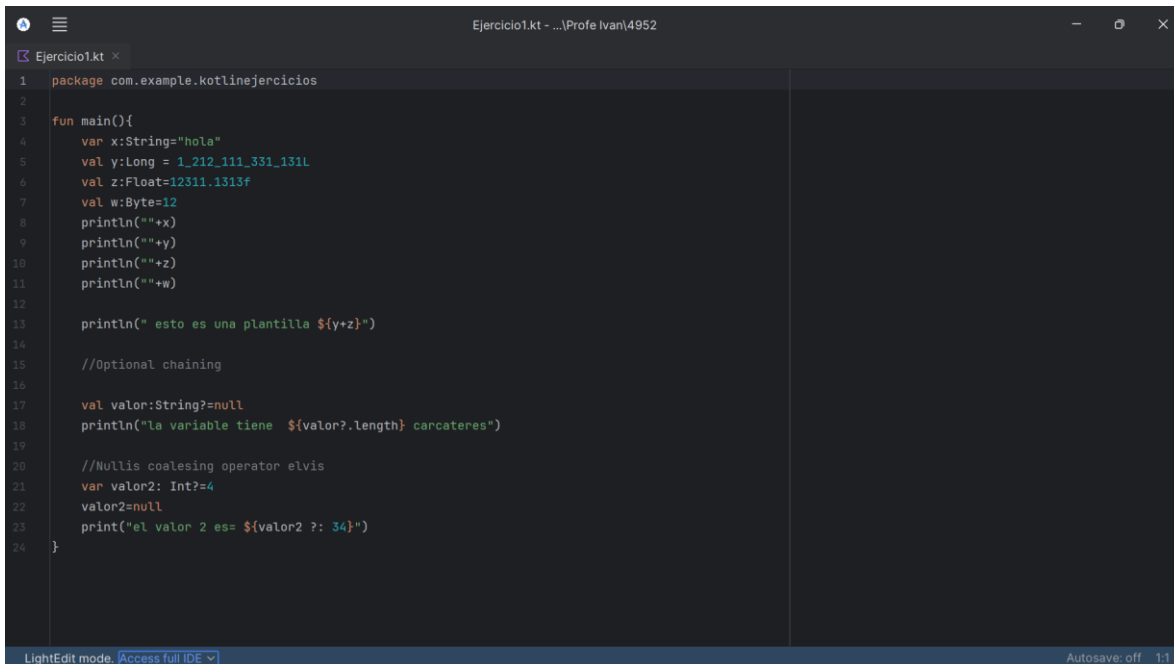
Un Layout define la estructura y disposición de los elementos visuales dentro de una pantalla de Android. Actúa como el “esqueleto” de la interfaz, indicando dónde y cómo se colocarán botones, textos, imágenes y otros componentes. Existen diferentes tipos de Layouts, como LinearLayout, RelativeLayout, ConstraintLayout y FrameLayout, cada uno con características específicas que facilitan la creación de interfaces adaptables y modernas. Los Layouts se suelen diseñar en XML, separando el diseño de la lógica de programación, y son esenciales para desarrollar aplicaciones visualmente coherentes y funcionales en distintos dispositivos y resoluciones.

EJERCICIOS KOTLIN

Ejercicio 01

Este fragmento de código muestra cómo declarar y utilizar distintos tipos de datos en Kotlin, incluyendo cadenas de texto, números enteros largos, números de punto flotante y números pequeños. Se imprimen los valores de estas variables en la consola utilizando la función `println`.

Además, se ejemplifica la interpolación de cadenas, que permite incluir operaciones directamente dentro de una cadena, como la suma de variables, para mostrar resultados de manera dinámica. También se destacan dos características importantes de Kotlin: el encadenamiento seguro (safe call), que permite acceder de manera segura a propiedades de variables que podrían ser nulas sin causar errores, y el operador Elvis (`?:`), que asigna un valor por defecto cuando una variable es nula.



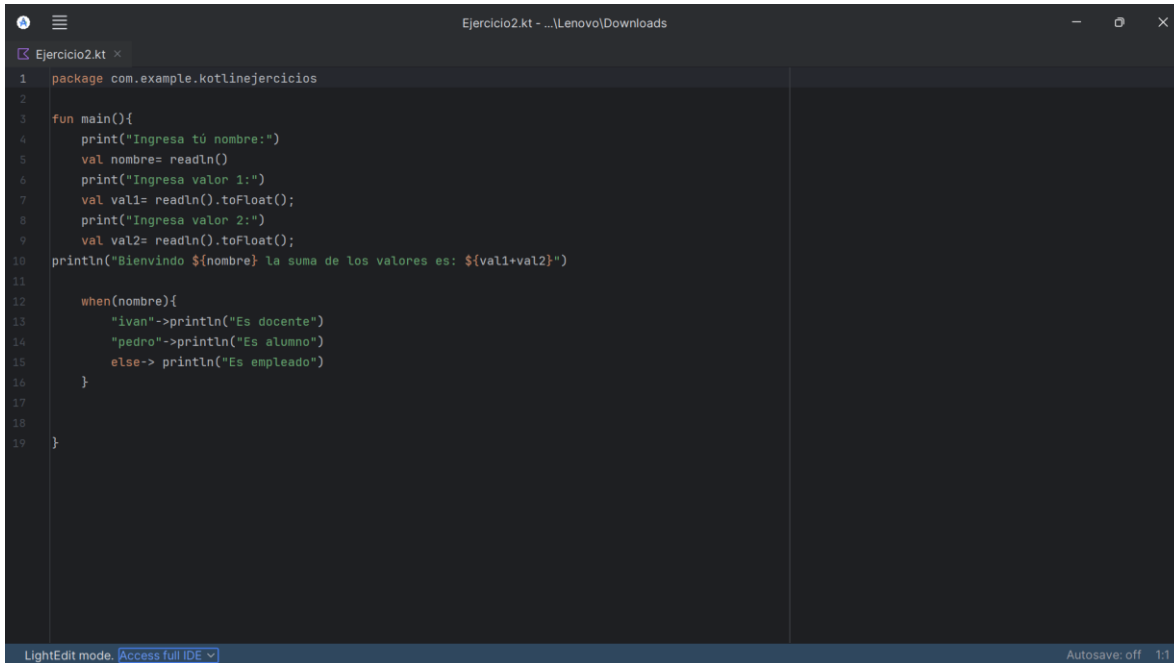
```
1 package com.example.kotlinejercicios
2
3 fun main(){
4     var x:String="hola"
5     val y:Long = 1_212_111_331_131L
6     val z:Float=12311.1313f
7     val w:Byte=12
8     println(""+x)
9     println(""+y)
10    println(""+z)
11    println(""+w)
12
13    println(" esto es una plantilla ${y+z}")
14
15    //Optional chaining
16
17    val valor:String?=null
18    println("la variable tiene ${valor?.length} caracteres")
19
20    //Nullis coalescing operator elvis
21    var valor2: Int?=4
22    valor2=null
23    print("el valor 2 es= ${valor2 ?: 34}")
24 }
```

Ejercicio 02

Este código utiliza la función `readln()` para recibir datos directamente desde la consola. Primero, solicita al usuario que ingrese su nombre, almacenándolo en la variable `nombre`. Luego, pide dos valores numéricos, los convierte a tipo `Float` y los guarda en las variables `val1` y `val2`.

Una vez obtenidos todos los datos, el programa imprime un mensaje de bienvenida que incluye el nombre del usuario y el resultado de la suma de `val1` y `val2`. Para

mostrar los resultados de manera dinámica junto con texto, se emplea la interpolación de cadenas (o plantillas de cadenas), permitiendo combinar operaciones y texto en una sola llamada a println.



```
1 package com.example.kotlinejercicios
2
3 fun main(){
4     print("Ingresa tu nombre:")
5     val nombre= readln()
6     print("Ingresa valor 1:")
7     val val1= readln().toFloat();
8     print("Ingresa valor 2:")
9     val val2= readln().toFloat();
10    println("Bienvido ${nombre} la suma de los valores es: ${val1+val2}")
11
12    when(nombre){
13        "ivan"->println("Es docente")
14        "pedro"->println("Es alumno")
15        else-> println("Es empleado")
16    }
17
18
19 }
```

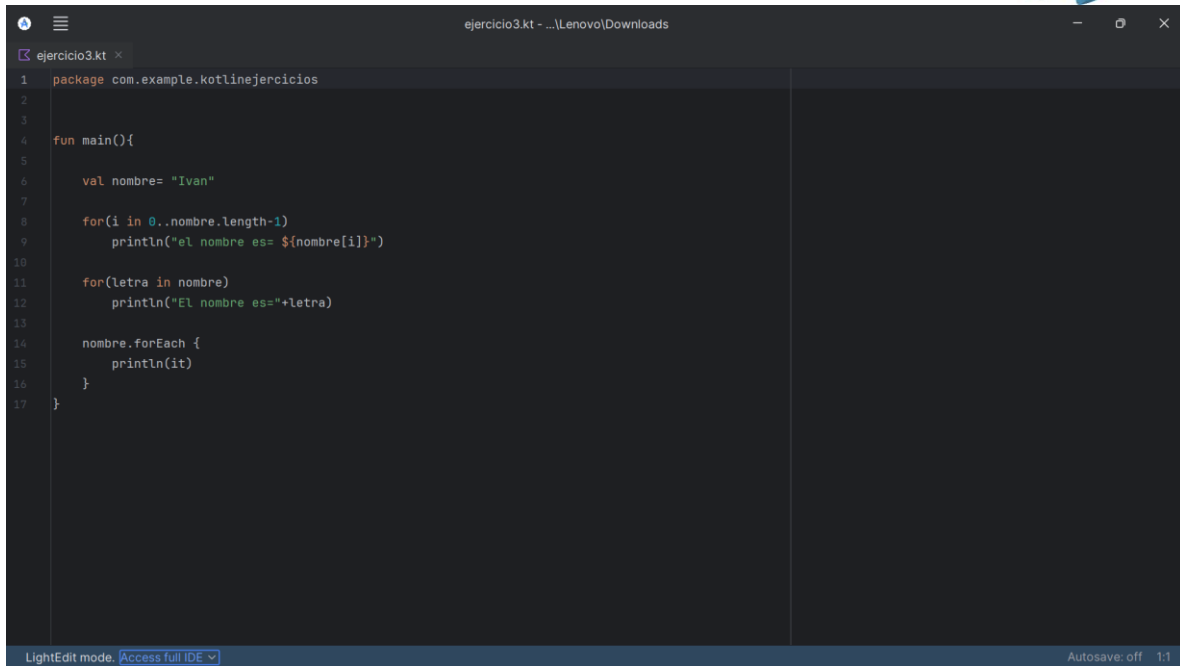
Ejercicio 03

El código define una variable llamada nombre con el valor de cadena "Ivan". A continuación, se muestran tres formas distintas de iterar sobre la cadena para procesar cada uno de sus caracteres de manera individual.

El primer método utiliza un bucle for con un rango de índices, recorriendo la cadena desde el índice 0 hasta el final (0..nombre.length-1). En cada iteración, se accede al carácter correspondiente mediante nombre[i].

El segundo método emplea un bucle for-each, que itera directamente sobre cada carácter de la cadena, eliminando la necesidad de trabajar con índices.

El tercer método utiliza la función forEach, una función de orden superior que acepta una expresión lambda. La lambda ejecuta un bloque de código para cada carácter de la cadena, proporcionando una forma concisa y funcional de realizar la misma tarea.



```
1 package com.example.kotlinejercicios
2
3
4 fun main(){
5
6     val nombre= "Ivan"
7
8     for(i in 0..nombre.length-1)
9         println("el nombre es= ${nombre[i]}")
10
11     for(letra in nombre)
12         println("El nombre es="+letra)
13
14     nombre.forEach {
15         println(it)
16     }
17 }
```

LightEdit mode. [Access full IDE](#) Autosave: off 1:1

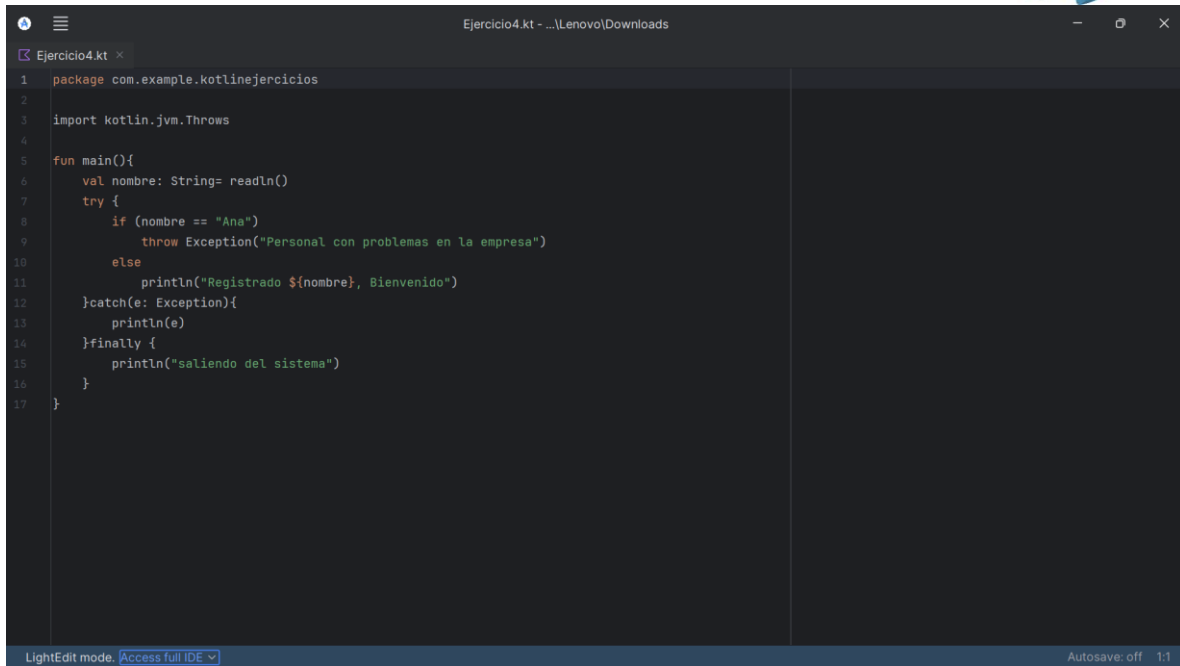
Ejercicio 04

El código solicita al usuario que ingrese un nombre en la consola mediante `readln()`. A continuación, se utiliza un bloque `try-catch-finally` para manejar posibles excepciones.

Dentro del bloque `try`, el nombre ingresado se compara con "Ana". Si el nombre coincide, se lanza una excepción con el mensaje "Personal con problemas en la empresa". Si el nombre es diferente, el programa muestra un mensaje de bienvenida confirmando el nombre del usuario.

Si se produce la excepción, el bloque `catch` la captura y muestra la información correspondiente en la consola. Finalmente, el bloque `finally` se ejecuta siempre, independientemente de si ocurrió o no una excepción, mostrando el mensaje "saliendo del sistema".

Este enfoque permite que el programa maneje de manera controlada tanto escenarios normales como situaciones excepcionales.

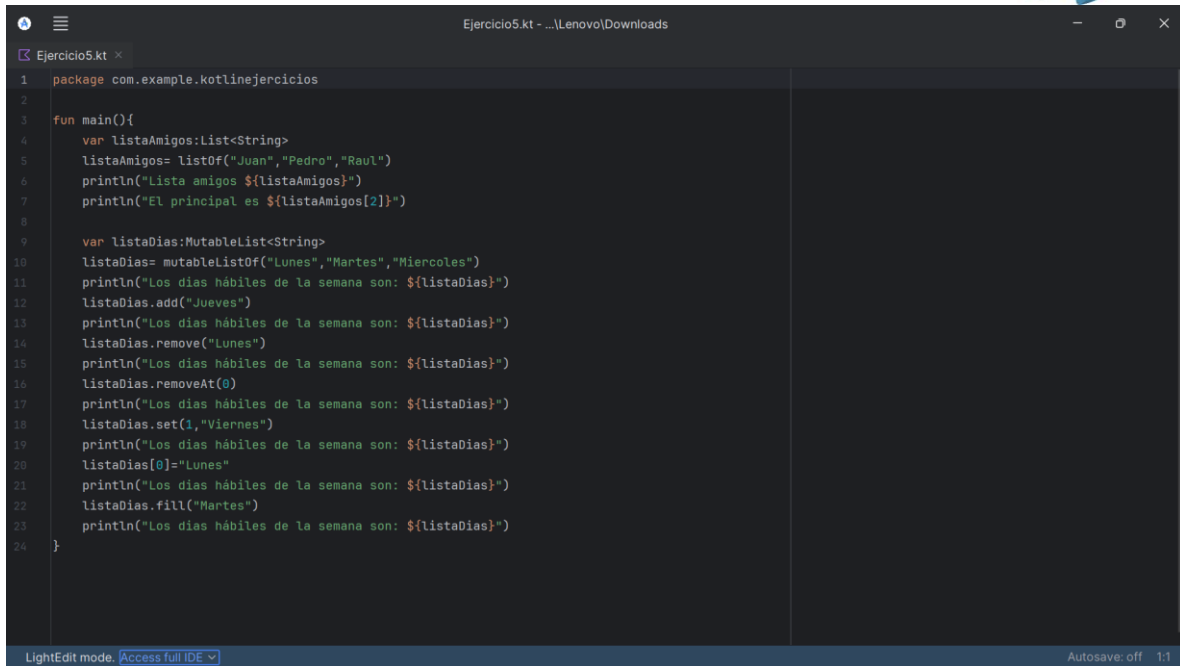


```
1 package com.example.kotlinejercicios
2
3 import kotlin.jvm.Throws
4
5 fun main(){
6     val nombre: String= readln()
7     try {
8         if (nombre == "Ana")
9             throw Exception("Personal con problemas en la empresa")
10        else
11            println("Registrado ${nombre}, Bienvenido")
12    }catch(e: Exception){
13        println(e)
14    }finally {
15        println("saliendo del sistema")
16    }
17 }
```

Ejercicio 05

En este código se trabajan dos tipos de listas: inmutables (List) y mutables (MutableList). Primero, se crea una lista inmutable llamada listaAmigos con los valores "Juan", "Pedro" y "Raul". Al ser inmutable, no se pueden agregar ni eliminar elementos después de su creación. El programa imprime la lista completa y accede a un elemento específico por su índice, en este caso el tercero (listaAmigos[2]), que corresponde a "Raul".

Posteriormente, se declara una lista mutable llamada listaDias con los valores "Lunes", "Martes" y "Miércoles". Al ser mutable, permite realizar operaciones como agregar, eliminar o modificar elementos. A lo largo del código, se aplican distintos cambios: primero se añade "Jueves", luego se elimina "Lunes", se elimina el elemento en la posición 0, se reemplaza el valor de la posición 1 por "Viernes" y, posteriormente, se cambia nuevamente el primer elemento a "Lunes". Finalmente, se utiliza el método fill("Martes") para sobrescribir todos los valores de la lista con "Martes".



```
1 package com.example.kotlinejercicios
2
3 fun main(){
4     var listaAmigos:List<String>
5     listaAmigos= listOf("Juan","Pedro","Raul")
6     println("Lista amigos ${listaAmigos}")
7     println("El principal es ${listaAmigos[2]}")
8
9     var listaDias:MutableList<String>
10    listaDias= mutableListOf("Lunes","Martes","Miercoles")
11    println("Los dias hábiles de la semana son: ${listaDias}")
12    listaDias.add("Jueves")
13    println("Los dias hábiles de la semana son: ${listaDias}")
14    listaDias.remove("Lunes")
15    println("Los dias hábiles de la semana son: ${listaDias}")
16    listaDias.removeAt(0)
17    println("Los dias hábiles de la semana son: ${listaDias}")
18    listaDias.set(1,"Viernes")
19    println("Los dias hábiles de la semana son: ${listaDias}")
20    listaDias[0]="Lunes"
21    println("Los dias hábiles de la semana son: ${listaDias}")
22    listaDias.fill("Martes")
23    println("Los dias hábiles de la semana son: ${listaDias}")
24 }
```

Ejercicio 06

El código muestra cómo trabajar con colecciones mutables e inmutables y realiza diversas operaciones como eliminación, mapeo, combinación, aplanamiento, filtrado y ordenamiento.

Manipulación de mapas (Map)

Se declara un mapa mutable llamado lista, que almacena productos como claves de tipo String y sus cantidades como valores de tipo Int. Inicialmente se agregan tres elementos: "refrescos", "enlatados" y "pan". Se imprime el mapa completo, se elimina la clave "pan" mostrando su valor, y luego se vuelve a imprimir para reflejar los cambios.

A continuación, se intenta incrementar los valores del mapa en uno. La función map se utiliza para mostrar cómo se podría transformar el contenido, aunque no altera el mapa original. En cambio, al usar forEach y reasignar los valores, sí se modifican directamente los elementos del mapa.

Operaciones con listas (List)

Se declaran dos listas inmutables, precios y productos. A la lista precios se le aplica un cálculo de IVA mediante la función map, pero al no guardar el resultado en una nueva variable, los valores originales permanecen intactos al imprimir la lista. Posteriormente, se utiliza el operador zip para combinar productos con precios, creando una nueva lista de pares que agrupa cada producto con su precio.

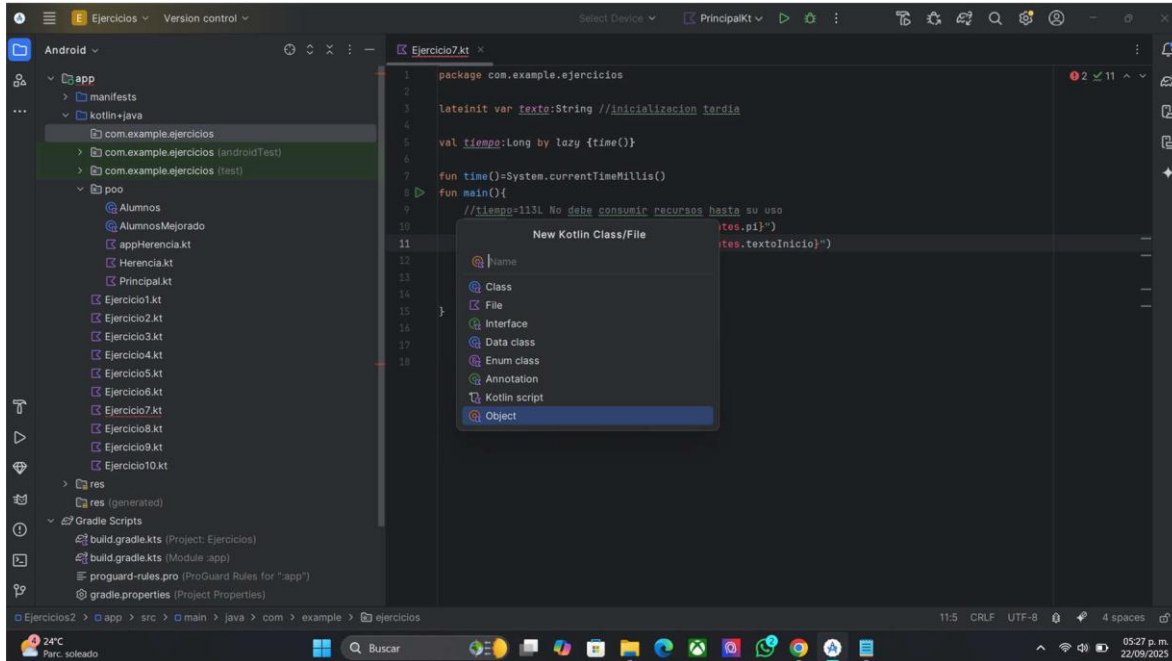
Finalmente, se muestra cómo aplanar listas de listas usando la función `flatten`, lo que permite convertir una colección anidada en una sola lista continua para su impresión o manipulación.

```
Ejercicio6.kt - ...\\Lenovo\\Downloads
Ejercicio6.kt x
1 package com.example.kotlinejercicios
2
3 fun main(){
4     var lista=mutableMapOf<String, Int>() //mutable
5     lista["refrescos"]=13
6     lista["enlatados"]=20
7     lista["pan"]=12
8
9     println(lista)
10    println("pan eliminado con valor de: ${lista.remove("pan")}")
11    println(lista)
12    lista.map {(k,v)->lista[k]=v+1}
13    lista.forEach{(k,v)->lista[k]=v+1}
14    println(lista)
15
16    val precios= listOf(200,3000,400) //No mutable
17    val productos = listOf("camisa","pantalon","blusa")
18    precios.map { it + (it * 0.16) }
19    //println("lista de precios con iva ${precios.map { it + (it * 0.16) }}")
20    println("lista de precios con iva $precios")
21    println("Productos: ${productos zip precios}")
22
23    var numeros= listOf( listOf(1,2,3), listOf(4,5,6), listOf(100,200,300))
24    println(numeros)
25    var n=numeros.flatten()
26    println(n)
27
28    var alumnos= listOf("Jose Juan","Pedro","Juan")
29    println(alumnos)
30    println(alumnos.filter { it.length <=4 })
31    println(alumnos.filter { it.contains("s") })
32    println("Indice de Juan: ${alumnos.indexOf("Juan")}")
33    println(alumnos.reversed())
34    println(alumnos.sorted())
35    println(alumnos.sortedBy { it.length })
36 }
```

```
Ejercicio6.kt - ...\\Lenovo\\Downloads
Ejercicio6.kt x
1 package com.example.kotlinejercicios
2
3 fun main(){
4     var lista=mutableMapOf<String, Int>() //mutable
5     lista["refrescos"]=13
6     lista["enlatados"]=20
7     lista["pan"]=12
8
9     println(lista)
10    println("pan eliminado con valor de: ${lista.remove("pan")}")
11    println(lista)
12    lista.map {(k,v)->lista[k]=v+1}
13    lista.forEach{(k,v)->lista[k]=v+1}
14    println(lista)
15
16    val precios= listOf(200,3000,400) //No mutable
17    val productos = listOf("camisa","pantalon","blusa")
18    precios.map { it + (it * 0.16) }
19    //println("lista de precios con iva ${precios.map { it + (it * 0.16) }}")
20    println("lista de precios con iva $precios")
21    println("Productos: ${productos zip precios}")
22
23    var numeros= listOf( listOf(1,2,3), listOf(4,5,6), listOf(100,200,300))
24    println(numeros)
25    var n=numeros.flatten()
26    println(n)
27
28    var alumnos= listOf("Jose Juan","Pedro","Juan")
29    println(alumnos)
30    println(alumnos.filter { it.length <=4 })
31    println(alumnos.filter { it.contains("s") })
32    println("Indice de Juan: ${alumnos.indexOf("Juan")}")
33    println(alumnos.reversed())
34    println(alumnos.sorted())
35    println(alumnos.sortedBy { it.length })
36 }
```

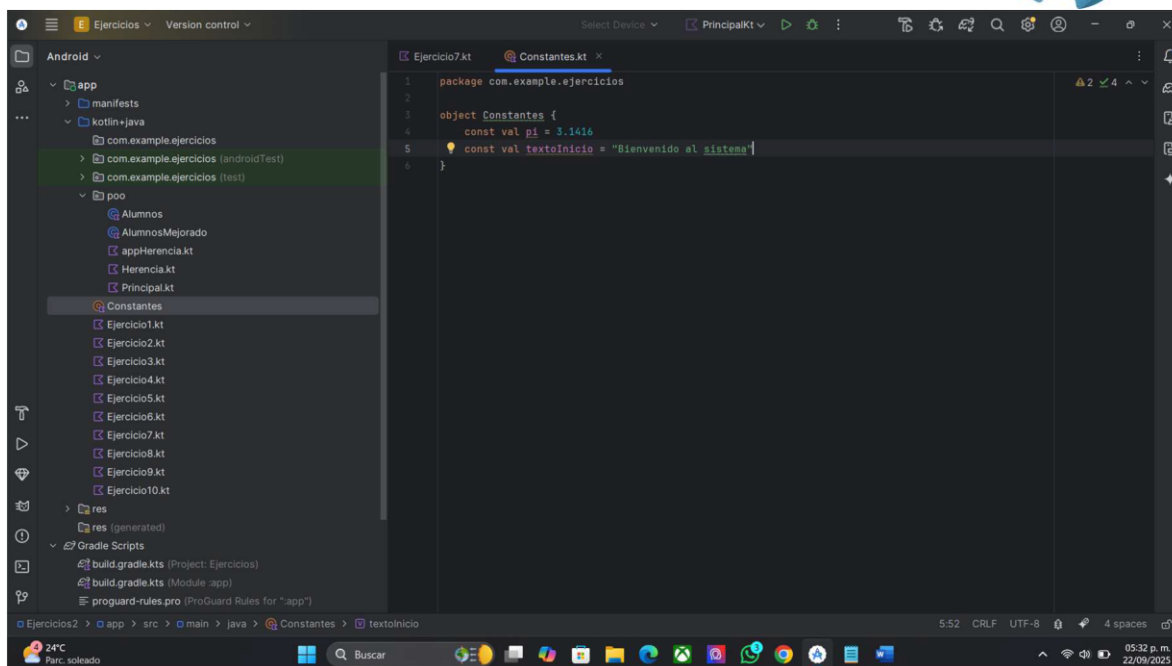
Ejercicio 07

Para este ejercicio, se realizó la creación de un objeto llamado Constantes, para realizarlo se siguió el procedimiento previamente visto y seleccionando la opción de “Object” para su creación.



Se crea un objeto con el nombre de “Constantes” usando el procedimiento de Object dentro de este objeto, vamos añadir los siguientes dos valores:

- PI: Un valor numérico para la constante PI.
- BIENVENIDA: Un texto para dar la bienvenida.

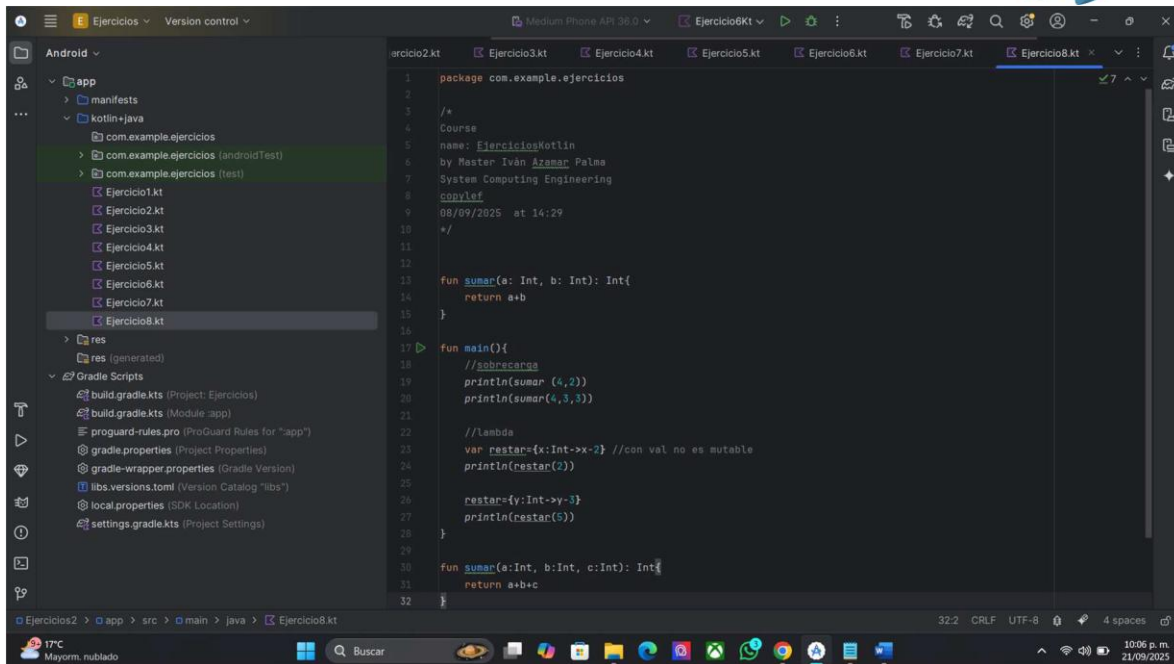


```
1 package com.example.ejercicios
2
3 object Constantes {
4     const val pi = 3.1416
5     const val textoInicio = "Bienvenido al sistema"
6 }
```

En este código se utilizan diferentes formas de inicializar variables en Kotlin. Por un lado, la palabra clave `lateinit` se emplea para declarar la variable `texto`, lo que permite inicializarla más adelante, antes de su primer uso, siendo útil cuando su valor depende de la entrada del usuario.

Por otro lado, la variable `tiempo: Long` se inicializa de manera diferida (*lazy*). Esto significa que su valor no se calcula hasta que se accede a ella por primera vez. En este caso, `tiempo` obtiene su valor al ejecutar la función `time()`, que devuelve la hora actual del sistema en milisegundos, evitando cálculos innecesarios.

Además, el objeto `Constantes` actúa como un contenedor de valores fijos. Contiene dos constantes: `pi`, con el valor 3.1416, y `textoInicio`, con la cadena "Bienvenido al sistema". Al usar `object`, este bloque se convierte en un singleton, lo que garantiza que solo exista una única instancia de `Constantes` accesible desde cualquier parte del programa.



```
1 package com.example.ejercicios
2
3 /*
4  * Course
5  * name: EjerciciosKotlin
6  * by Master Iván Azamar Palma
7  * System Computing Engineering
8  * copyleft
9  * 08/09/2025 at 14:29
10  */
11
12
13 fun sumar(a: Int, b: Int): Int {
14     return a+b
15 }
16
17 fun main() {
18     //sobrecarga
19     println(sumar(4,2))
20     println(sumar(4,3,3))
21
22     //Lambda
23     var restar={x:Int->x-2} //con val no es mutable
24     println(restar(2))
25
26     restar={y:Int->y-3}
27     println(restar(5))
28 }
29
30 fun sumar(a:Int, b:Int, c:Int): Int {
31     return a+b+c
32 }
```

Ejercicio 08

El código ilustra el uso de funciones, sobrecarga de funciones y expresiones lambda.

Funciones y Sobrecarga

Se define primero la función sumar (a: Int, b: Int), que recibe dos enteros y devuelve su suma. Posteriormente, se define otra función con el mismo nombre sumar (a: Int, c: Int, d: Int), que recibe tres enteros. Este es un ejemplo de sobrecarga de funciones, que permite usar el mismo nombre para distintas funciones siempre que se diferencien por la cantidad o el tipo de parámetros. En la función principal (main), se llaman ambas versiones de sumar, mostrando la suma de dos números (4 + 2) y de tres números (4 + 3 + 3).

Expresiones Lambda

El código también demuestra el uso de lambdas, que son funciones anónimas. Se asigna a la variable restar una lambda que recibe un entero y le resta 2. Al llamarla con el valor 2, el resultado es 0. Luego, restar se reasigna a otra lambda que resta 3 al número recibido; al llamarla con 5, el resultado es 2. Esto muestra la flexibilidad de las lambdas, que pueden ser reasignadas y reutilizadas para distintas operaciones.

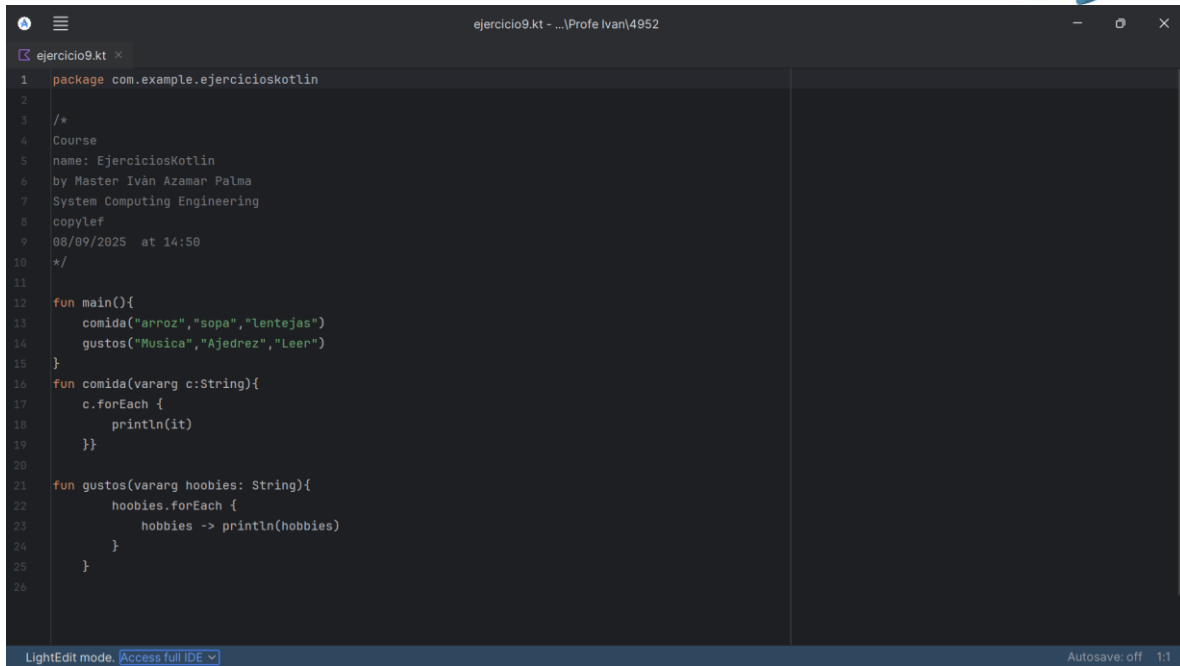

```
ejercicio8.kt - ...\\Profe Ivan\\4952
ejercicio8.kt x
1 package com.example.ejercicioskotlin
2
3 /*
4  * Course
5  * name: EjerciciosKotlin
6  * by Master Iván Azamar Palma
7  * System Computing Engineering
8  * copyleft
9  * 08/09/2025 at 14:29
10  */
11
12
13 fun sumar(a: Int, b: Int): Int{
14     return a+b
15 }
16
17 fun main(){
18     //sobrecarga
19     println(sumar (4,2))
20     println(sumar(4,3,3))
21
22     //Lambda
23     var restar={x:Int->x-2} //con val no es mutable
24     println(restar(2))
25
26     restar={y:Int->y-3}
27     println(restar(5))
28 }
LightEdit mode. Access full IDE v Autosave: off 1:1
```

Ejercicio 09

El código muestra cómo utilizar la palabra clave `vararg` para manejar un número variable de parámetros en una función.

En la función `main`, se realizan llamadas a dos funciones: `comida` y `gustos`. Ambas aceptan múltiples argumentos en una sola llamada gracias a `vararg`.

La función `comida` (`vararg c: String`) recibe una cantidad indefinida de cadenas y las recorre con `forEach` para imprimir cada una. De manera similar, la función `gustos` (`vararg hobbies: String`) recibe varios textos y también utiliza `forEach` para mostrar cada gusto o pasatiempo del usuario.



```
1 package com.example.ejercicioskotlin
2
3 /*
4  * Course
5  * name: EjerciciosKotlin
6  * by Master Iván Azamar Palma
7  * System Computing Engineering
8  * copyleft
9  * 08/09/2025 at 14:50
10 */
11
12 fun main(){
13     comida("arroz", "sopa", "lentejas")
14     gustos("Musica", "Ajedrez", "Leer")
15 }
16 fun comida(vararg c:String){
17     c.forEach {
18         println(it)
19     }
20 }
21 fun gustos(vararg hobbies: String){
22     hobbies.forEach {
23         hobbies -> println(hobbies)
24     }
25 }
26 }
```

Ejercicio 10

El programa define dos funciones para mostrar la diferencia entre una operación asíncrona no controlada y una que utiliza un callback para devolver resultados de manera segura.

Función operación (Sincronización inapropiada)

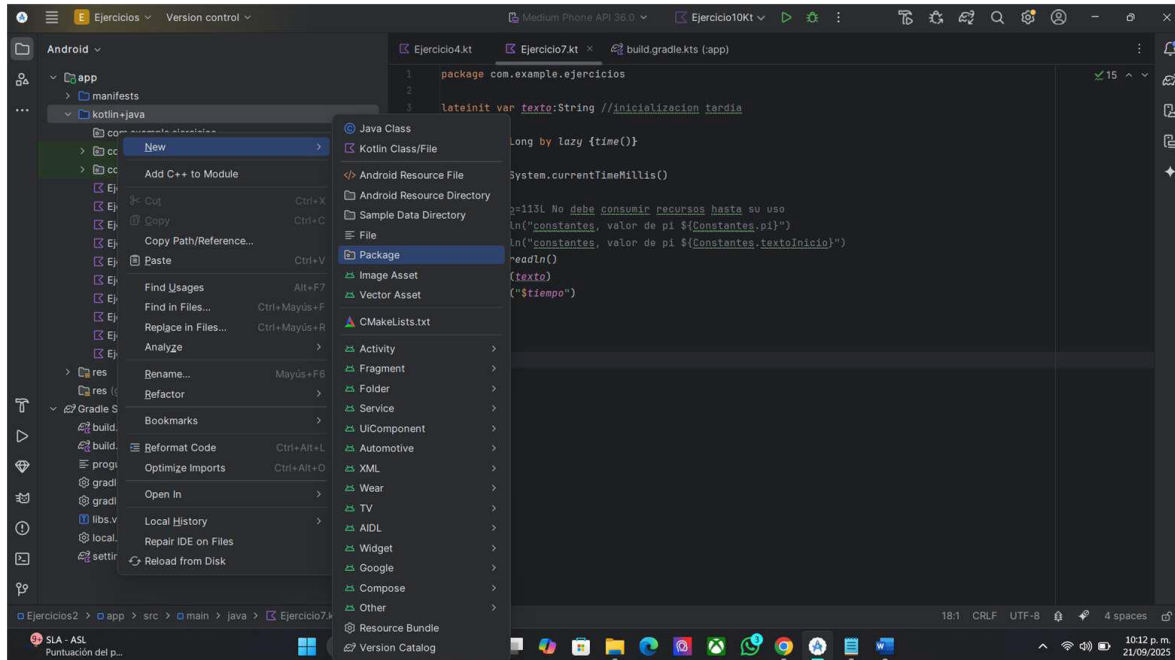
La función operación (a, b) crea un nuevo hilo (Thread) para realizar un cálculo. Dentro del hilo, se simula una pausa de dos segundos para representar una tarea pesada y luego se calcula $a + b * 2$, guardando el resultado en la variable z. Sin embargo, la función devuelve z de inmediato, sin esperar a que el hilo termine su ejecución. Esto provoca un problema de sincronización, ya que el resultado siempre será 0, porque el hilo aún no ha completado el cálculo al momento de la devolución.

Función operacion2 (Uso de callback)

La función operacion2(a, b, onResult) resuelve el problema usando un callback, que es una función de orden superior pasada como parámetro. Esta función también crea un hilo, simula la espera de dos segundos y realiza el cálculo. En lugar de devolver el valor inmediatamente, llama a onResult pasando el resultado del cálculo, garantizando que la operación asíncrona complete su ejecución antes de entregar el resultado. Esto demuestra cómo Kotlin permite manejar funciones de orden superior de manera efectiva, asegurando la correcta transferencia de valores en operaciones asíncronas.

```
ejercicio10.kt - ...Profe Ivan\4952
ejercicio10.kt x
1 package com.example.ejercicioskotlin
2
3 /*
4  * Course
5  * name: EjerciciosKotlin
6  * by Master Iván Azamar Palma
7  * System Computing Engineering
8  * copyleft
9  * 08/09/2025 at 15:04
10  */
11 fun main(){
12     println(operacion(4,3))
13     //funcion de orden superior
14     val result=operacion2(2,4, {
15         println(it)
16     })
17
18     val result2=operacion2(2,4, {
19         z->println(z)
20     })
21
22     val result3=operacion2(3,4){ //solo cuando callback es al final, los resultados no salen ordenados
23         z->println(z)
24     }
25 }
26
27 fun operacion(a:Int, b:Int): Int{
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
26
```

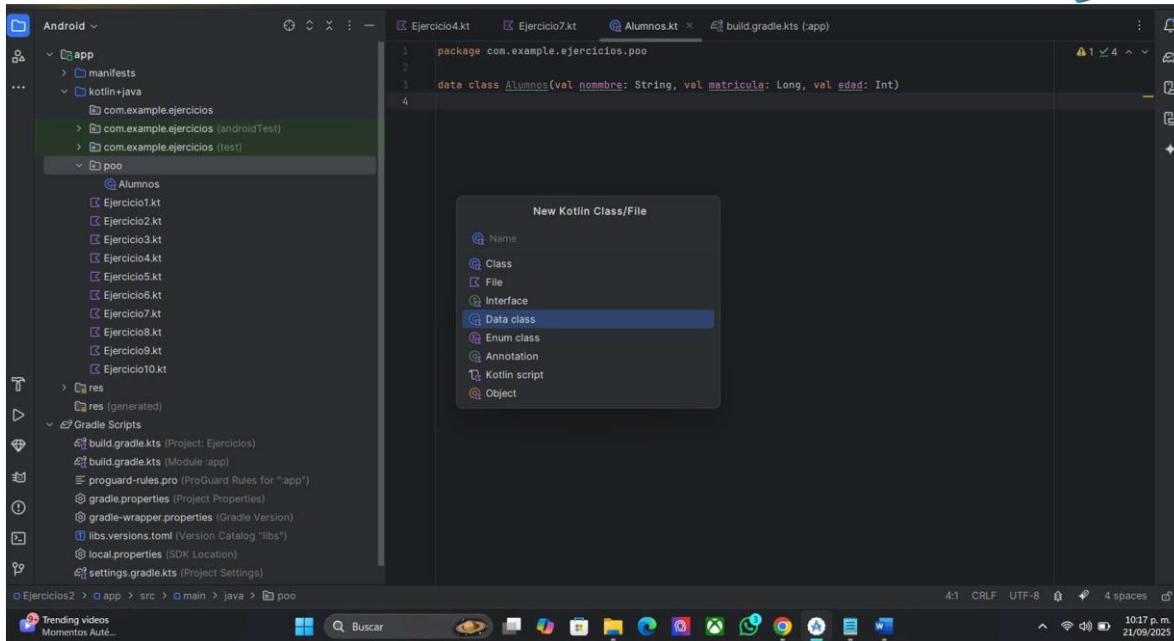
De esta manera, se genera un paquete que servirá para organizar de manera adecuada las clases relacionadas con la programación orientada a objetos (POO), lo cual facilita el mantenimiento, la escalabilidad y la comprensión del proyecto, siguiendo las buenas prácticas de desarrollo en Android y Kotlin.



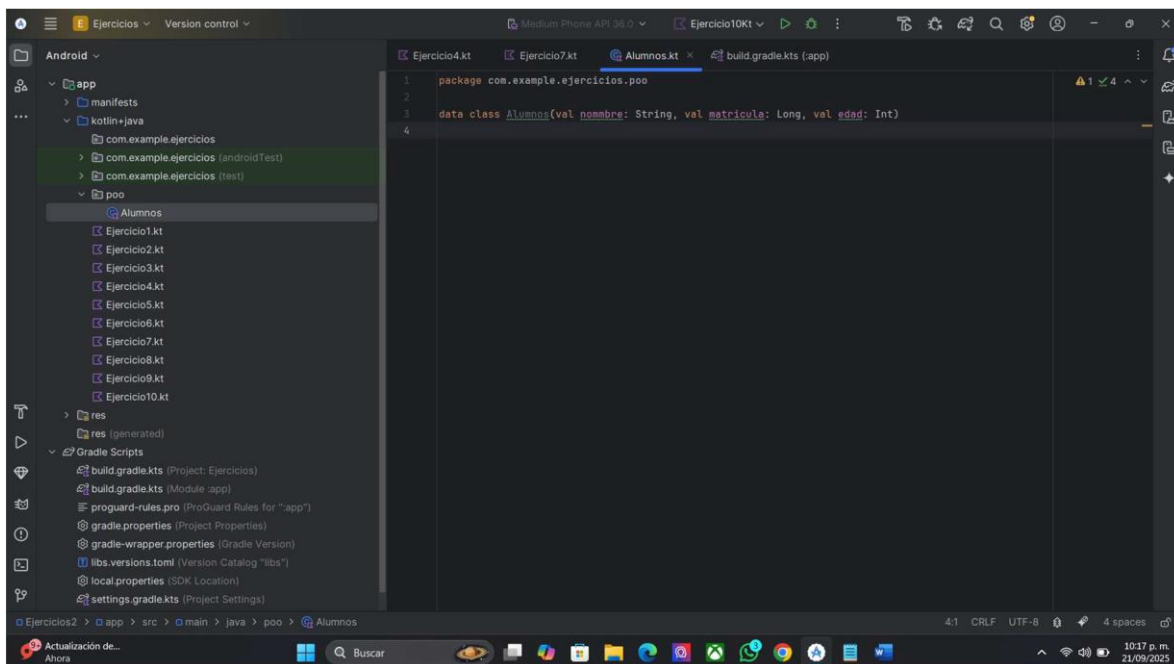
Data Class

Para crear una Data Class en Kotlin, el procedimiento es prácticamente el mismo que al crear una clase normal. Primero, asegúrate de estar dentro de la carpeta poo. Luego, haz clic derecho sobre ella, selecciona New y, en las opciones mostradas, elige Kotlin Class/File. Después, marca el tipo como Data Class y coloca el nombre que desees siguiendo las reglas de Kotlin.

Las Data Classes son muy prácticas porque permiten manejar información de forma clara y ordenada. Además, Kotlin genera automáticamente métodos como toString(), equals(), hashCode() y copy(), evitando escribir código repetitivo y facilitando el trabajo con objetos.



Una data class se emplea para guardar información y en este caso incluye tres propiedades: nombre, matrícula y edad. En Kotlin, las data classes generan de forma automática métodos como `toString()`, `equals()` y `copy()`, lo que facilita su manejo y reduce código innecesario. En otras palabras, este archivo representa un modelo de datos sencillo para un estudiante, pensado únicamente para almacenar y administrar su información.



Class Alumno Mejorado

Esta clase representa un modelo más completo de un alumno, integrando principios de la programación orientada a objetos (POO) como la encapsulación y el uso de constructores. Incluye un constructor principal con valores por defecto y un constructor secundario que inicializa los atributos cursos y nota. La variable *cursos* está declarada con `lateinit`, lo que permite asignarle un valor más adelante, mientras que *nota* es privada, restringiendo su acceso desde fuera de la clase y reforzando la encapsulación. La clase también define dos métodos:

- `getDatos()`: devuelve un texto con la información principal del alumno.
- `toString()`: sobrescribe la representación en texto para mostrar todos los atributos, incluyendo *cursos* y *nota*.

```

package com.example.ejercicios.poo

class AlumnosMejorado(val nombre: String="", val matricula: Long=0, val edad: Byte=0,) {
    lateinit var cursos: String
    private var nota: Int=0

    constructor(nombre: String, matricula: Long, edad: Byte, cursos: String, nota: Int): this(nombre, matricula, edad) {
        this.cursos = cursos
        this.nota = nota
    }

    // Método para consultar todos los datos
    fun getDatos(): String {
        return "Nombre: $nombre, Matricula: $matricula, Edad: $edad, Persona: $cursos"
    }

    override fun toString(): String {
        return "AlumnosMejorado(nombre='$nombre', matricula=$matricula, edad=$edad, cursos='$cursos', nota=$nota)"
    }
}
    
```

Principal

El programa ejemplifica el uso de dos tipos de clases en Kotlin: una data class (*Alumnos*) y una clase convencional con funcionalidades extendidas (*AlumnosMejorado*).

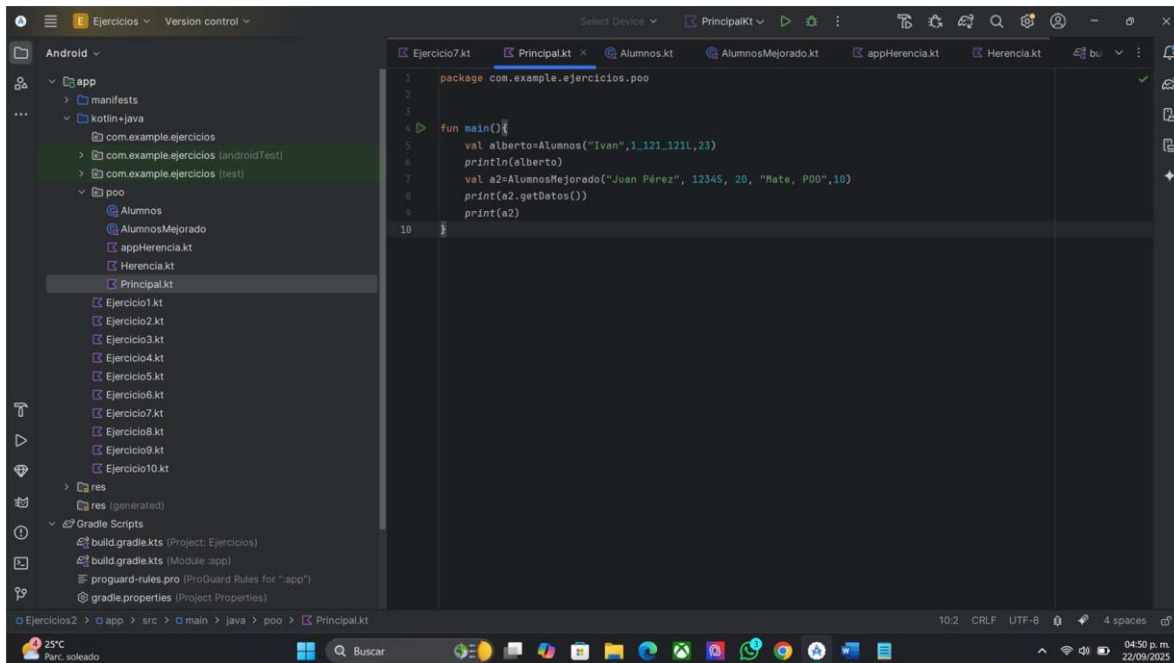
Uso de Data Class

Se crea un objeto llamado *alberto* de tipo *Alumnos*, al cual se le asignan los valores "Ivan", matrícula 1121121 y edad 23. Al imprimirlo con `println(alberto)`, Kotlin ejecuta automáticamente el método `toString()` propio de las *data class*, mostrando los atributos del objeto de forma legible y estructurada, sin necesidad de código adicional.

Uso de Clase Normal con Métodos

Posteriormente, se instancia un objeto *a2* de tipo *AlumnosMejorado*, inicializado con datos más completos: nombre "Juan Pérez", matrícula 12345, edad 20, cursos "Mate, POO" y nota 10.

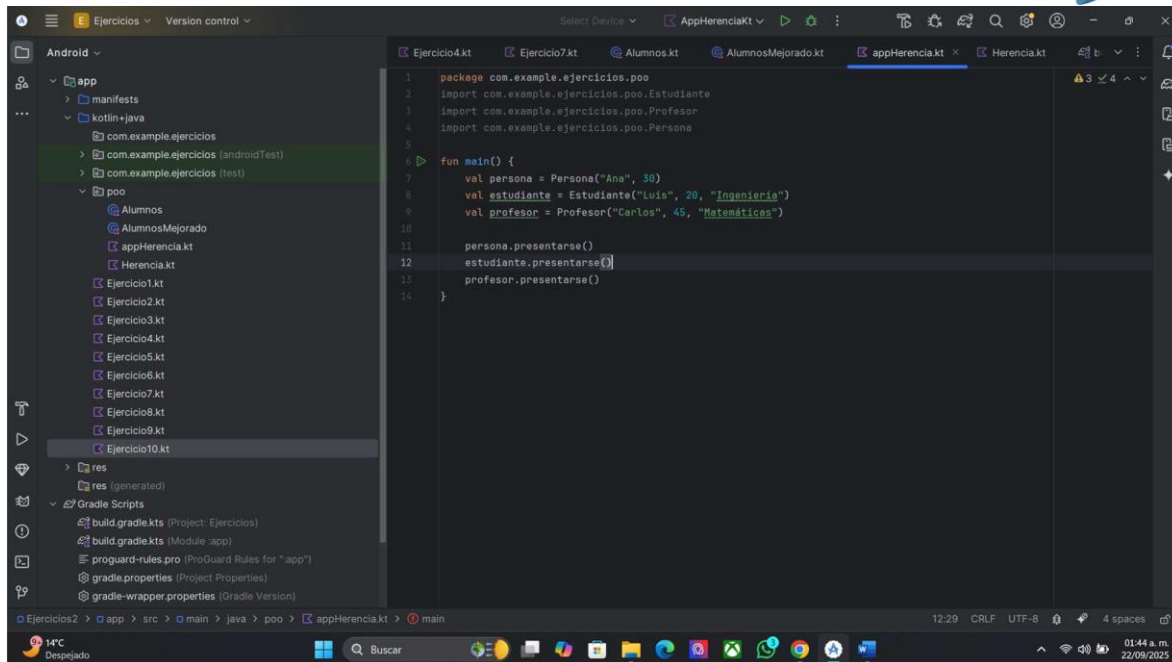
Al llamar al método *a2.getDatos()*, se obtiene una cadena con la información principal del alumno. Finalmente, al imprimir directamente el objeto *a2*, se invoca el método *toString()* sobrescrito en la clase, mostrando una descripción personalizada de todos los atributos, incluyendo cursos y nota.



```
1 package com.example.ejercicios.poo
2
3
4 fun main() {
5     val alberto=Alumnos("Ivan",1_121_121,21)
6     println(alberto)
7     val a2=AlumnosMejorado("Juan Pérez", 12345, 20, "Mate, POO",10)
8     print(a2.getDatos())
9     print(a2)
10 }
```

APP Herencia

En la función *main* se instancian tres clases: una *Persona* llamada Ana, un *Estudiante* llamado Luis que cursa la carrera de Ingeniería y un *Profesor* llamado Carlos que imparte Matemáticas. Posteriormente, se ejecuta el método *presentarse()* en cada uno de los objetos. Como este método ha sido sobrescrito en cada subclase, la salida es diferente en cada caso, a pesar de que la invocación sea exactamente la misma. Esto constituye un ejemplo claro de polimorfismo, uno de los principios fundamentales de la programación orientada a objetos.



```
1 package com.example.ejercicios.poe
2 import com.example.ejercicios.poe.Estudiante
3 import com.example.ejercicios.poe.Profesor
4 import com.example.ejercicios.poe.Persona
5
6 fun main() {
7     val persona = Persona("Ana", 30)
8     val estudiante = Estudiante("Luis", 20, "Ingeniería")
9     val profesor = Profesor("Carlos", 45, "Matemáticas")
10
11     persona.presentarse()
12     estudiante.presentarse()
13     profesor.presentarse()
14 }
```

Herencia

Este archivo, junto con appHerencia, ejemplifica el uso de la herencia y el polimorfismo.

En primer lugar, se declara la clase base *Persona*, que contiene las propiedades *nombre* y *edad*, además de un método *presentarse()* que muestra un mensaje simple con los datos de la persona. Posteriormente, se definen dos clases que extienden de *Persona*: *Estudiante* y *Profesor*. Ambas redefinen el método *presentarse()*. En el caso de *Estudiante*, se añade la propiedad *carrera* y se personaliza el mensaje para incluirla, mientras que *Profesor* incorpora la propiedad *materia* y modifica la presentación para señalar qué materia imparte.


```

1 package com.example.ejercicios.poo
2
3
4 open class Persona(val nombre: String, val edad: Int) {
5     open fun presentarse() {
6         println("Hola, soy $nombre y tengo $edad años.")
7     }
8 }
9
10 // Clase derivada
11 class Estudiante(nombre: String, edad: Int, val carrera: String) : Persona(nombre, edad) {
12     override fun presentarse() {
13         println("Hola, soy $nombre, tengo $edad años y estudio $carrera.")
14     }
15 }
16
17 // Clase derivada
18 class Profesor(nombre: String, edad: Int, val materia: String) : Persona(nombre, edad) {
19     override fun presentarse() {
20         println("Hola, soy el profesor $nombre, enseño $materia.")
21     }
22 }
23
  
```

IMC

Introducción

El propósito del presente trabajo es explicar el desarrollo de una aplicación móvil que tiene la finalidad de calcular el IMS (Índice de Masa Saludable) de las personas, con la capacidad de poder calcular el IMS de diversas personas. El proceso comienza desde el diseño hecho a mano, para luego digitalizarlo utilizando la herramienta Figma y, posteriormente, implementarlo en Android Studio. El desarrollo del código de esta aplicación toma en cuenta toda la información que se ha visto en clase, integrando conceptos fundamentales de programación en Kotlin.

En los primeros ejercicios se ejemplifica la manipulación de distintos tipos de datos, incluyendo cadenas de texto, números enteros y flotantes, así como el uso de interpolación de cadenas para combinar texto y operaciones directamente dentro de una expresión. Se introducen también conceptos clave de Kotlin como el encadenamiento opcional y el operador Elvis, que permiten manejar de manera segura valores nulos y asignar valores por defecto cuando es necesario.

Se exploran técnicas de entrada de datos desde la consola con la función `readln()`, mostrando cómo capturar información del usuario y realizar operaciones como la suma de valores ingresados, integrando la interpolación de cadenas para presentar resultados de manera dinámica. Además, se ilustran distintos métodos para iterar sobre cadenas, incluyendo bucles `for` con rangos de índices, bucles `for-each` y la función `forEach` con expresiones lambda, permitiendo procesar caracteres individualmente de forma eficiente y concisa.

El manejo de errores se aborda mediante el uso de bloques try-catch-finally, mostrando cómo controlar escenarios excepcionales, como la detección de ciertos nombres específicos, y garantizando que se ejecuten acciones finales independientemente de si ocurre una excepción. Esto permite desarrollar aplicaciones más robustas y seguras frente a entradas inesperadas del usuario.

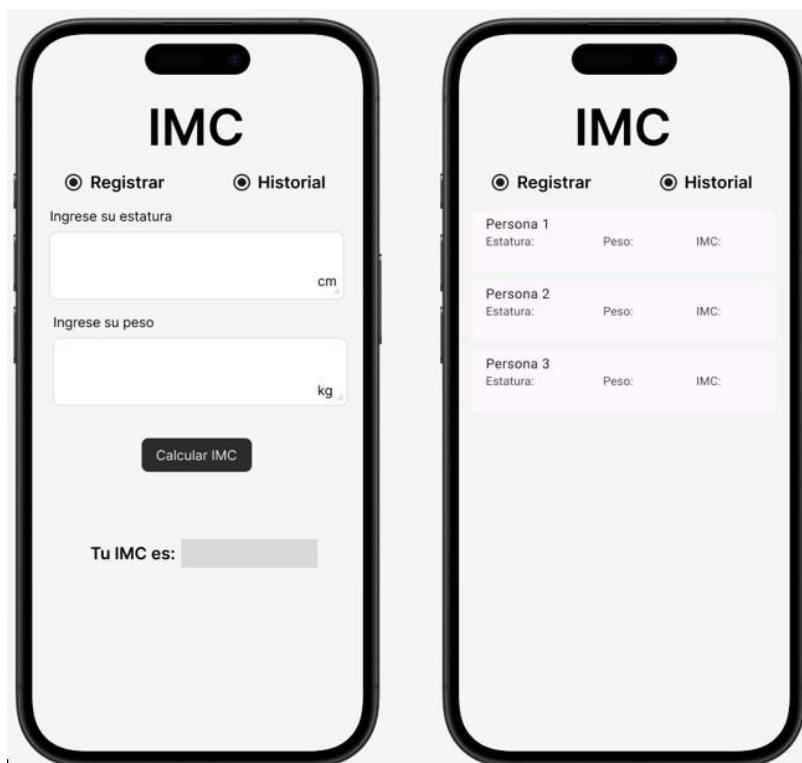
El trabajo con colecciones se ejemplifica mediante listas inmutables y mutables, mostrando cómo agregar, eliminar, reemplazar y llenar elementos de manera controlada. Asimismo, se abordan operaciones avanzadas con mapas y listas, incluyendo la combinación de colecciones, aplanamiento de listas de listas, filtrado, ordenamiento y manipulación de valores mediante funciones map y forEach. También se introduce la palabra clave vararg para manejar un número variable de parámetros en funciones, optimizando la reutilización de código y la flexibilidad de la aplicación.

Por último, se presentan conceptos de programación asíncrona y uso de callbacks, demostrando cómo manejar operaciones que requieren tiempo de ejecución, asegurando que los resultados se entreguen correctamente solo cuando la tarea se completa. Esto es esencial para aplicaciones móviles que realizan cálculos o consultas que podrían demorarse, como el cálculo de IMS para múltiples personas de manera simultánea.

En conjunto, este trabajo integra de manera práctica y didáctica los fundamentos de Kotlin vistos en clase, desde tipos de datos básicos, entrada y salida de información, manejo de errores, colecciones y funciones avanzadas, hasta programación asíncrona. Todo ello con el objetivo de desarrollar una aplicación móvil funcional y eficiente que cumpla con la finalidad de calcular el IMS de manera precisa y amigable para el usuario.

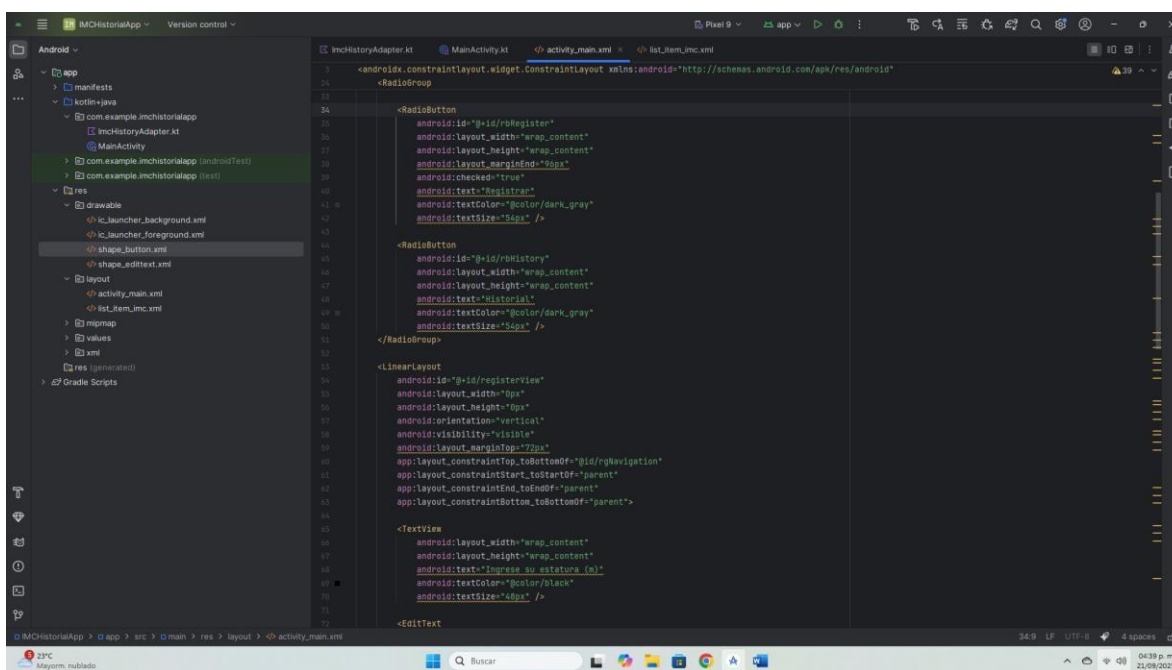
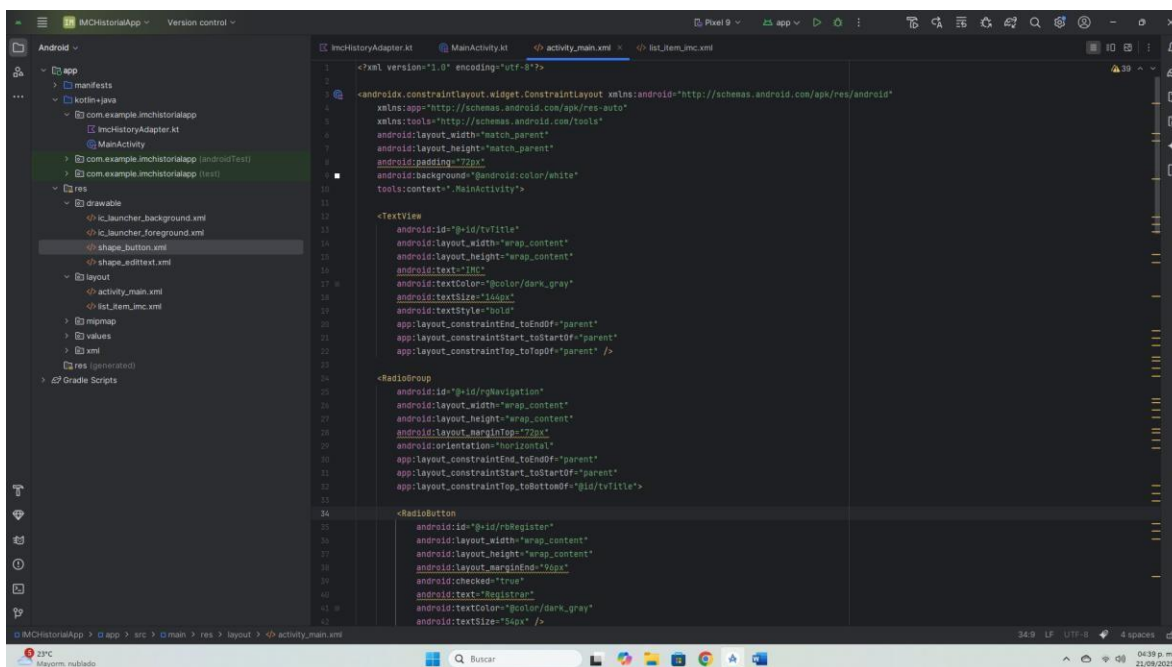
Desarrollo

Como primer paso tomaremos el diseño hecho en figma para posteriormente hacerlo en Android Studio.



En la primera pantalla se muestran 2 radiobuttons los cuales permiten registrar el IMC y para ver el historial. Ahí mismo se muestran 2 cajas de texto las cuales permiten ingresar la estatura y el peso y posteriormente hay un botón el cual permite calcular el IMC, y debajo de este se hay una caja de texto la cual muestra el IMC calculado.

Y en la segunda pantalla se muestra el historial de IMC anteriormente calculados.



```

1  <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      <LinearLayout
3          <EditText
4              android:id="@+id/etStature"
5              android:layout_width="match_parent"
6              android:layout_height="150px"
7              android:layout_marginTop="30px"
8              android:background="@drawable/shape_edittext"
9              android:hint="kg"
1             android:inputType="numberDecimal"
11             android:paddingHorizontal="40px" />
12
13         <TextView
14             android:layout_width="wrap_content"
15             android:layout_height="wrap_content"
16             android:layout_marginTop="40px"
17             android:text="Ingresa su peso (kg)"
18             android:textColor="@color/black"
19             android:textSize="18px" />
20
21         <EditText
22             android:id="@+id/etWeight"
23             android:layout_width="match_parent"
24             android:layout_height="150px"
25             android:layout_marginTop="30px"
26             android:background="@drawable/shape_edittext"
27             android:hint="kg"
28             android:inputType="numberDecimal"
29             android:paddingHorizontal="40px" />
30
31         <View
32             android:layout_width="0px"
33             android:layout_height="0px"
34             android:layout_weight="1" />
35
36         <Button
37             android:id="@+id/btnCalculate"
38             android:layout_width="match_parent"
39             android:layout_height="150px"
40             android:background="@drawable/shape_button"
41             android:text="Calcular IMC"

```

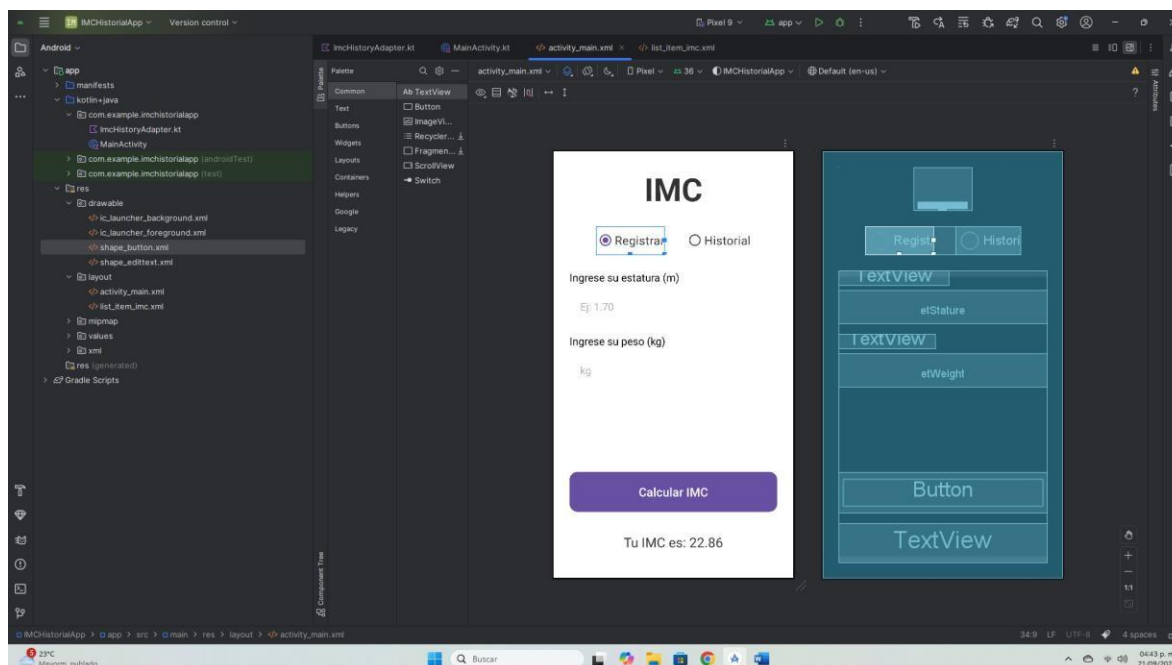
```

1  <Button
2      android:id="@+id/btnCalculate"
3      android:layout_width="match_parent"
4      android:layout_height="150px"
5      android:background="@drawable/shape_button"
6      android:text="Calcular IMC"
7      android:textColor="@color/white"
8      android:textSize="18px" />
9
10 <TextView
11     android:id="@+id/tvResult"
12     android:layout_width="match_parent"
13     android:layout_height="wrap_content"
14     android:layout_marginTop="40px"
15     android:gravity="center"
16     android:padding="40px"
17     android:textColor="@color/dark_gray"
18     android:textSize="18px"
19     tools:text="Tu IMC es: 22.86" />
20
21 </LinearLayout>
22
23 <androidx.recyclerview.widget.RecyclerView
24     android:id="@+id/historyView"
25     android:layout_width="0px"
26     android:layout_height="0px"
27     android:visibility="gone"
28     android:layout_marginTop="70px"
29     app:layout_constraintTop_toBottomOf="@id/rghNavigation"
30     app:layout_constraintStart_toStartOf="parent"
31     app:layout_constraintEnd_toEndOf="parent"
32     app:layout_constraintBottom_toBottomOf="parent"
33     tools:listitem="@layout/list_item_inc" />
34
35 </androidx.constraintlayout.widget.ConstraintLayout>

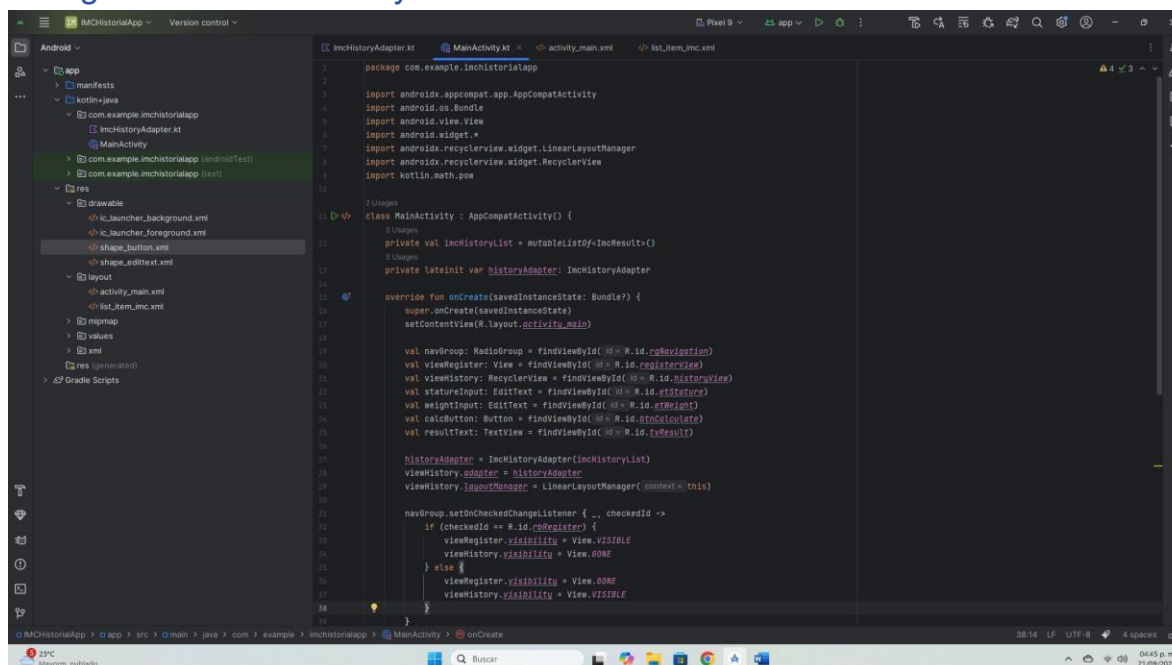
```

En las capturas de pantalla se muestran los códigos de diseño de los widgets.

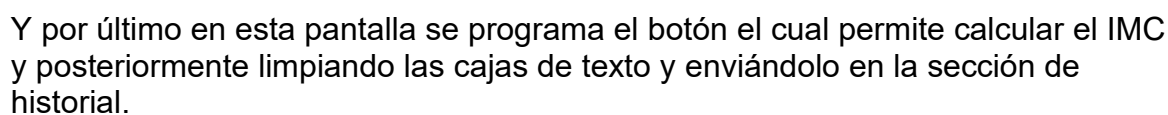
Plantilla De Diseño De Android Studio.



Código De La Mainactivity



En la primera parte de nuestro código se programó un evento el cual permite cambiar de sección cuando se toca el botón de historial se muestra esta y si se toca el botón de registrar se muestra esa pantalla.



The screenshot shows the Android Studio IDE with the following details:

- Project Structure (Left Sidebar):**
 - app
 - manifests
 - kotlin-java
 - com.example.imchitorialapp
 - IMCHistorialAdapter.kt
 - MainActivity
 - com.example.imchitorialapp (androidTest)
 - com.example.imchitorialapp (test)
 - res
 - drawable
 - ic_launcher_background.xml
 - ic_launcher_foreground.xml
 - shape_button.xml
 - shape_edittext.xml
 - layout
 - activity_main.xml
 - list_item_img.xml
 - mipmap
 - values
 - xml
 - res (generated)
 - Gradle Scripts

- Main Editor (XML Code):**

```

1 <?xml version="1.0" encoding="utf-8"?>
2
3 <androidx.cardview.widget.CardView xmlns:android="http://schemas.android.com/apk/res/android"
4     xmlns:app="http://schemas.android.com/apk/res-auto"
5     xmlns:tools="http://schemas.android.com/tools"
6     android:layout_width="match_parent"
7     android:layout_height="wrap_content"
8     android:layout_marginHorizontal="48px"
9     android:layout_marginVertical="26px"
10    app:cardBackgroundColor="@color/light_gray"
11    app:cardCornerRadius="16px"
12    app:cardElevation="8px">
13
14    <androidx.constraintlayout.widget.ConstraintLayout
15        android:layout_width="match_parent"
16        android:layout_height="wrap_content"
17        android:padding="48px">
18
19        <TextView
20            android:id="@+id/tvPersonIdentifier"
21            android:layout_width="wrap_content"
22            android:layout_height="wrap_content"
23            android:textColor="@color/dark_gray"
24            android:textSize="36px"
25            android:textStyle="bold"
26            app:layout_constraintStart_toStartOf="parent"
27            app:layout_constraintTop_toTopOf="parent"
28            tools:text="Persona 1" />
29
30        <TextView
31            android:id="@+id/tvStature"
32            android:layout_width="wrap_content"
33            android:layout_height="wrap_content"
34            android:layout_marginTop="26px"
35            android:textColor="@color/medium_gray"
36            android:textSize="36px"
37            app:layout_constraintStart_toStartOf="parent"
38            app:layout_constraintTop_toBottomOf="@id/tvPersonIdentifier"
39            tools:text="Stature: 1.78 m" />
40
41        <TextView
42            android:id="@+id/tvWeight"

```
- Bottom Status Bar:**
- IMCHistorialApp > app > src > main > res > layout > activity_main.xml
- 23°C
- Q Buscar
- 11 ORLF UTF-8 4 spaces

```

<androidx.cardview.widget.CardView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_marginTop="20px"
    android:textColor="@color/medium_gray"
    android:textSize="14sp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/tvPersonIdentifier"
    tools:text="Estatura: 1.70 m" />

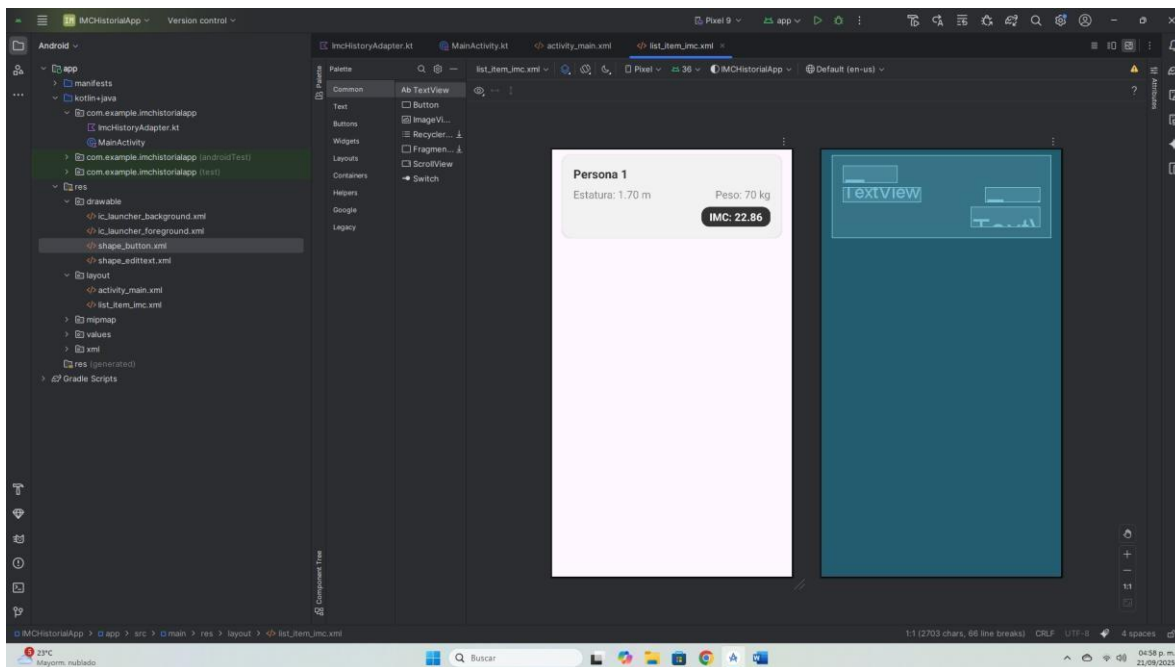
    <TextView
        android:id="@+id/tvWeight"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="@color/medium_gray"
        android:textSize="14sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="@id/tvStature"
        tools:text="Peso: 70 kg" />

    <TextView
        android:id="@+id/tvIMCResult"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="20px"
        android:background="@drawable/shape_button"
        android:paddingHorizontal="10px"
        android:paddingVertical="10px"
        android:textColor="@color/white"
        android:textSize="14sp"
        android:textStyle="bold"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toBottomOf="@id/tvWeight"
        tools:text="IMC: 22.86" />

</androidx.constraintlayout.widget.ConstraintLayout>
</androidx.cardview.widget.CardView>
    
```

Dentro de este código XML se muestra el diseño de esta pantalla

Diseño Visual De La Pantalla Historial.



Código De La Pantalla De Historial.

```

1 package com.example.inchistorialapp
2 import android.view.LayoutInflater
3 import android.view.View
4 import android.view.ViewGroup
5 import android.widget.TextView
6 import androidx.recyclerview.widget.RecyclerView
7
8 data class IncResult(val stature: Double, val weight: Double, val bmi: Double)
9
10 class IncHistoryAdapter(private val historyList: List<IncResult>) :
11     RecyclerView.Adapter<IncHistoryAdapter.IncViewHolder>() {
12
13     4 Usages
14     class IncViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
15         1 Usage
16         val personIdentifier: TextView = itemView.findViewById<?>(<Id> R.id.tvPersonIdentifier)
17         1 Usage
18         val stature: TextView = itemView.findViewById<?>(<Id> R.id.tvStature)
19         1 Usage
20         val weight: TextView = itemView.findViewById<?>(<Id> R.id.tvWeight)
21         1 Usage
22         val incResult: TextView = itemView.findViewById<?>(<Id> R.id.tvIncResult)
23     }
24
25     override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): IncViewHolder {
26         val view = LayoutInflater.from(context).inflate(R.layout.item_inc, parent, false)
27         return IncViewHolder(view)
28     }
29
30     override fun onBindViewHolder(holder: IncViewHolder, position: Int) {
31         val currentItem = historyList[position]
32
33         holder.personIdentifier.text = "Persona $<position> + 1"
34         holder.stature.text = "Estatura: $<currentItem.stature> m"
35         holder.weight.text = "Peso: $<currentItem.weight> kg"
36         holder.incResult.text = "IMC: $<String.format>(<format> '%.2f', currentItem.bmi)"
37     }
38
39     override fun getItemCount() = historyList.size
40 }

```

Este código se encarga de recibir los datos registrados y posteriormente mostrarlos en las cajas de texto de estatura, pero y IMC.

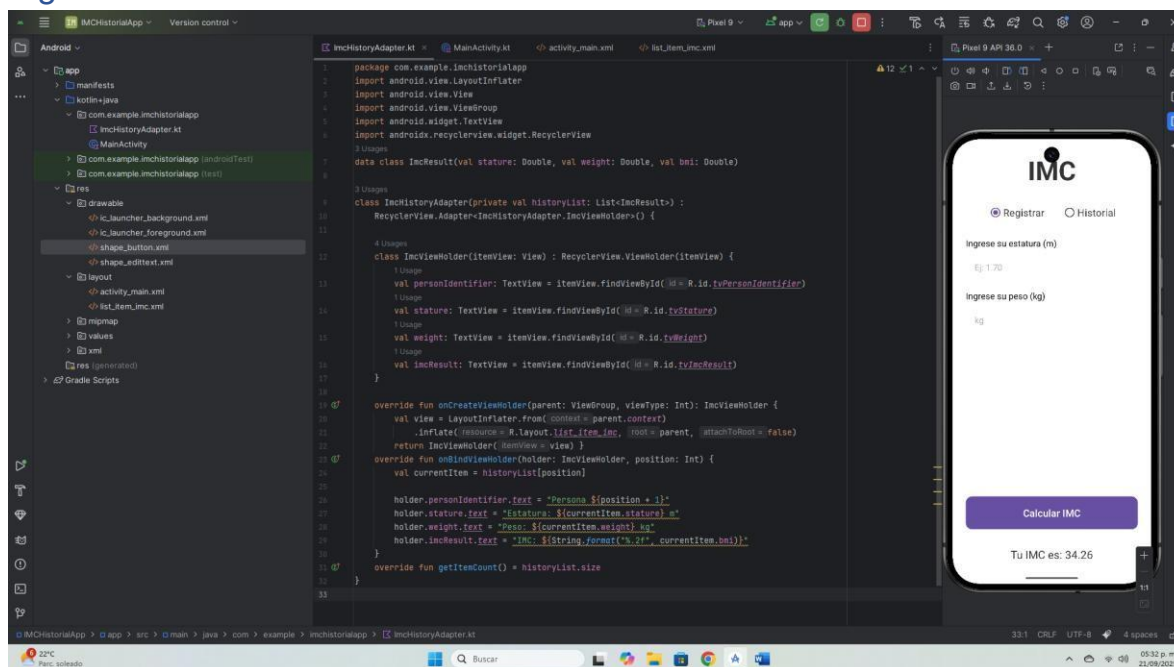
Funcionamiento De La Aplicación.

```

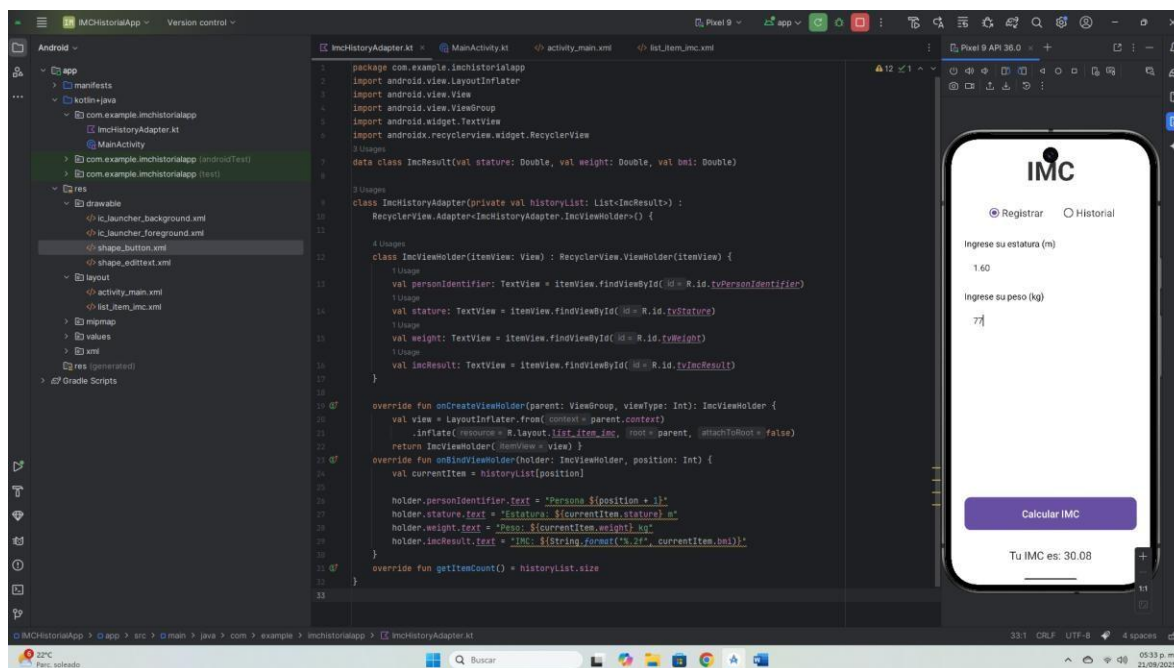
1 package com.example.inchistorialapp
2 import android.view.LayoutInflater
3 import android.view.View
4 import android.view.ViewGroup
5 import android.widget.TextView
6 import androidx.recyclerview.widget.RecyclerView
7
8 data class IncResult(val stature: Double, val weight: Double, val bmi: Double)
9
10 class IncHistoryAdapter(private val historyList: List<IncResult>) :
11     RecyclerView.Adapter<IncHistoryAdapter.IncViewHolder>() {
12
13     4 Usages
14     class IncViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
15         1 Usage
16         val personIdentifier: TextView = itemView.findViewById<?>(<Id> R.id.tvPersonIdentifier)
17         1 Usage
18         val stature: TextView = itemView.findViewById<?>(<Id> R.id.tvStature)
19         1 Usage
20         val weight: TextView = itemView.findViewById<?>(<Id> R.id.tvWeight)
21         1 Usage
22         val incResult: TextView = itemView.findViewById<?>(<Id> R.id.tvIncResult)
23     }
24
25     override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): IncViewHolder {
26         val view = LayoutInflater.from(context).inflate(R.layout.item_inc, parent, false)
27         return IncViewHolder(view)
28     }
29
30     override fun onBindViewHolder(holder: IncViewHolder, position: Int) {
31         val currentItem = historyList[position]
32
33         holder.personIdentifier.text = "Persona $<position> + 1"
34         holder.stature.text = "Estatura: $<currentItem.stature> m"
35         holder.weight.text = "Peso: $<currentItem.weight> kg"
36         holder.incResult.text = "IMC: $<String.format>(<format> '%.2f', currentItem.bmi)"
37     }
38
39     override fun getItemCount() = historyList.size
40 }

```

Ingreso De Datos De La Primera Persona.

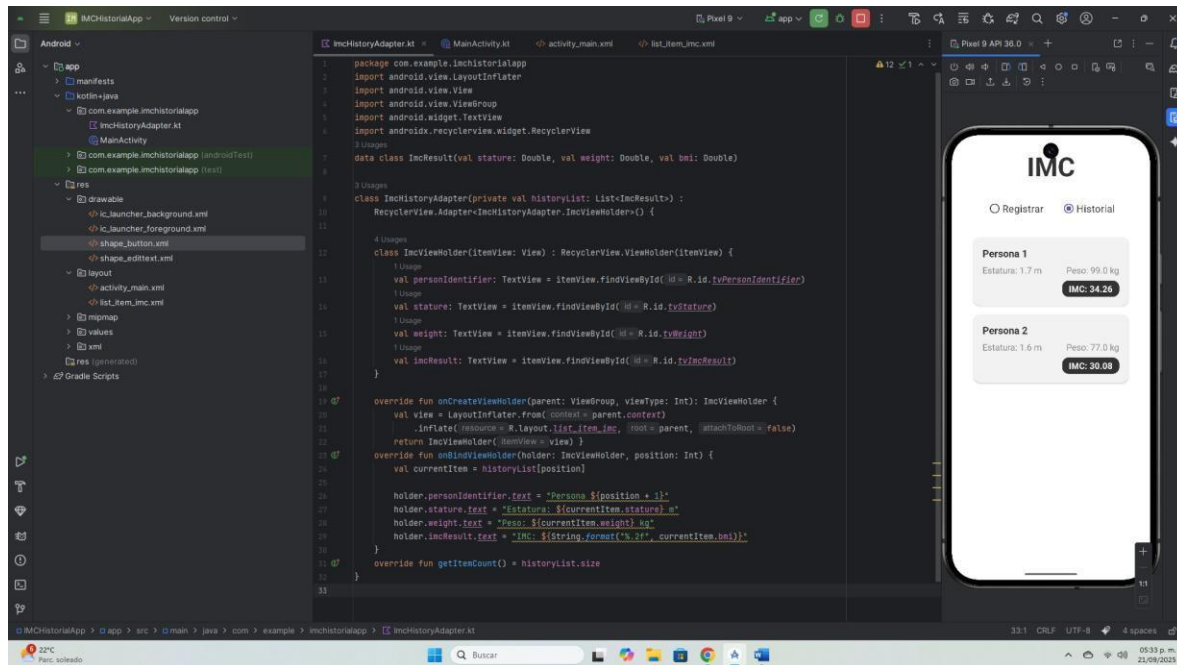


IMC Calculado



Datos de la segunda persona y IMC calculado

Verificación Del Funcionamiento El Historial.



CONCLUSION

El desarrollo de la aplicación de cálculo del Índice de Masa Corporal (IMC) permitió integrar de manera práctica los principales conceptos vistos en clase. A través del uso de Android Studio como entorno de desarrollo, del SDK 35 de Android y del lenguaje Kotlin, fue posible implementar una aplicación funcional que combina la lógica de programación con el diseño visual definido en XML, siguiendo buenas prácticas de desarrollo móvil.

La aplicación evidencia la importancia de los layouts y widgets en la creación de interfaces intuitivas, así como la relevancia de las activities y sus métodos del ciclo de vida, como onCreate, para garantizar la correcta inicialización de los componentes de la interfaz. Además, permite comprender cómo interactúan distintos elementos de la aplicación, desde botones y campos de texto hasta listas y mensajes dinámicos que se actualizan según la entrada del usuario.

Se aplicaron estructuras modernas de Kotlin, como val, var, funciones (fun) y la sobrescritura de funciones (override fun), integrando conceptos de programación funcional mediante lambdas, forEach y el manejo de colecciones mutables e inmutables. También se implementaron mecanismos de manejo de errores mediante bloques try-catch-finally, garantizando que la aplicación pueda responder de manera controlada a entradas inesperadas o errores de ejecución.

En cuanto a la interacción con el usuario, se aplicaron técnicas para capturar y procesar datos desde la consola y la interfaz, usando interpolación de cadenas para mostrar resultados dinámicos y claros, y se experimentó con conceptos de programación asíncrona y callbacks, demostrando cómo Kotlin permite ejecutar operaciones de manera segura y eficiente incluso en procesos que requieren tiempo de cálculo.

Además, se integraron conceptos de diseño reusable y modular, como el uso de objetos singleton para constantes y el manejo de funciones con vararg para parámetros variables, favoreciendo la escalabilidad y organización del código. Estos elementos reflejan la aplicación práctica de la teoría de programación, mostrando cómo pequeñas decisiones de diseño y estructura influyen directamente en la eficiencia y mantenibilidad del software.

En conclusión, este proyecto no solo permitió emplear los conocimientos vistos en clase, sino que también demostró cómo los fundamentos de la programación en Android y Kotlin se traducen en aplicaciones reales, funcionales y amigables para el usuario. El resultado final es una aplicación sencilla, clara y funcional, que integra de manera coherente teoría y práctica, evidenciando el valor del aprendizaje activo y la experimentación en el proceso educativo.