



GOBIERNO DE  
MÉXICO

EDUCACIÓN  
SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO  
NACIONAL DE MÉXICO



## TECNOLOGICO NACIONAL DE MEXICO INSTITUTO TECNOLOGICO DE CIUDAD MADERO

Carrera: Ingeniería en Sistemas Computacionales.

Materia: Programación Nativa para Móviles.

Maestro: Jorge Peralta Escobar.

Integrantes del equipo:

Karla Denisse Cruz Solís #21070310

Yahir Osvaldo Valero Hernández #21070330

Grupo: A

Hora: 09:00 – 10:00

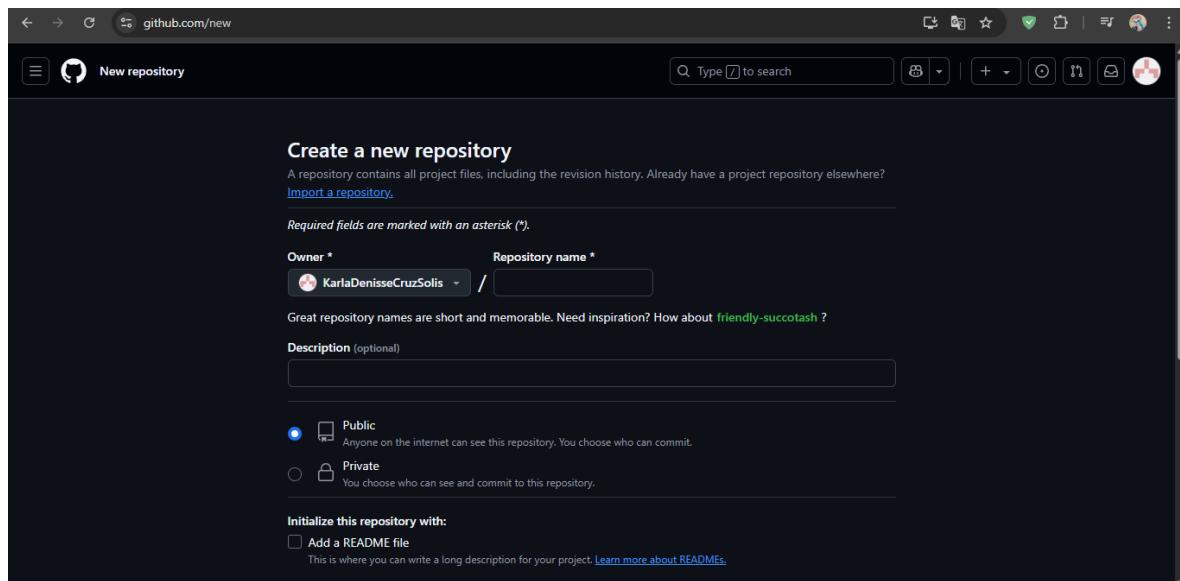
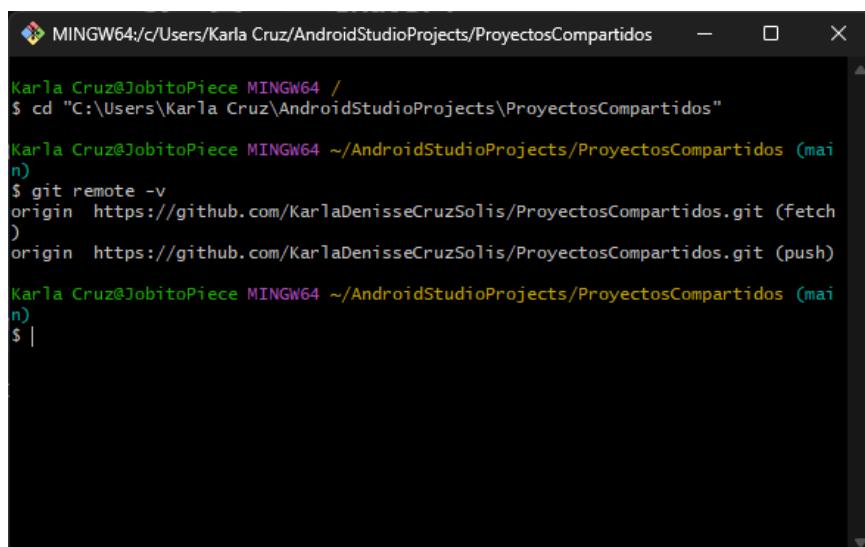
Semestre: Enero - Junio 2025.

## Documentación de Control de Versiones GIT y otros ejercicios

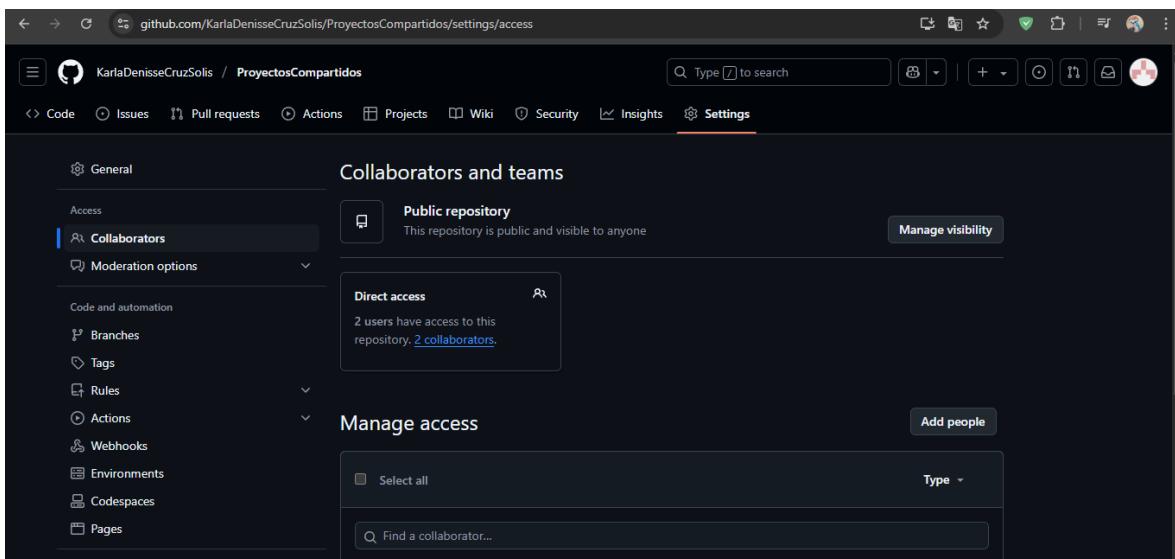
Este documento explica el proceso completo que se llevó a cabo para configurar y administrar el control de versiones mediante la sincronización de Git y GitHub en Android Studio, Visual Studio Code, IntelliJ o lo que vayamos a utilizar.

Se detalla desde la creación de la estructura del proyecto, la subida al repositorio remoto, la creación de ramas para trabajo colaborativo, hasta la resolución de conflictos y sincronización final en la rama principal (main).

Primeramente, comenzamos con la creación del repositorio en github, en nuestro caso llamado “ProyectosCompartidos” (el cual inicializamos localmente con git init y le agregamos el repositorio remoto con git remote add origin <https://github.com/KarlaDenisseCruzSolis/ProyectosCompartidos.git>, para después revisar que esté conectado correctamente con git remote -v:

Después agregamos a un compañero como colaborador en GitHub, que para permitir que nuestro compañero también pudiera realizar cambios directamente en el repositorio, se configuraron permisos de colaboración en GitHub, para lo cual en la configuración del repositorio se agregó al compañero como colaborador mediante la siguiente ruta: **Settings → Collaborators → Add collaborator**

A screenshot of a web browser displaying the GitHub repository settings for 'ProyectosCompartidos'. The 'Settings' tab is active. On the left, a sidebar shows 'Access' with 'Collaborators' selected. The main area is titled 'Collaborators and teams' and shows a 'Public repository' status with a note: 'This repository is public and visible to anyone'. Below this, under 'Direct access', it says '2 users have access to this repository. 2 collaborators.' A 'Manage visibility' button is present. At the bottom, there's a 'Manage access' section with a 'Select all' checkbox and a 'Type' dropdown, along with a search bar containing 'Find a collaborator...'. The overall theme is dark.

Se le envió la invitación y nuestro compañero la aceptó para que ambos pudiéramos trabajar en el mismo repositorio. Esto permitió que ambos pudiéramos crear ramas, realizar cambios y sincronizarlos directamente con el repositorio remoto, de lo cual hablaremos más adelante.

Por otra parte, para poder subir nuestros proyectos que teníamos en Android Studio a GitHub tuvimos que mover la carpeta del proyecto desde el explorador de archivos, esto desde la ubicación original a la carpeta del repositorio local. Por ejemplo:

Se movió la carpeta HappyBirthday desde:

C:\Users\Karla Cruz\AndroidStudioProjects\HappyBirthday

A:

C:\Users\Karla Cruz\AndroidStudioProjects\ProyectosCompartidos

Esto permitió que el proyecto estuviera dentro del repositorio local para facilitar el control de versiones.

Luego de mover el proyecto, se abrieron los comandos de Git Bash para registrar y confirmar los cambios en el repositorio:

1. Nos posicionamos en la carpeta del repositorio:  
`cd "C:/Users/Karla Cruz/AndroidStudioProjects/ProyectosCompartidos"`
2. Agregamos los cambios al área de preparación:  
`git add .`
3. Confirmamos los cambios con un mensaje descriptivo:  
`git commit -m "Agregando HappyBirthday al repositorio"`
4. Subimos los cambios al repositorio remoto:  
`git push origin main`

Esto permitió que el proyecto HappyBirthday apareciera en el repositorio en GitHub. Y así con cada uno de los proyectos que ya teníamos creados en Android en ese tiempo.

Por otra parte, para la creación de ramas para el trabajo colaborativo, para que cada uno trabajara de manera independiente; se crearon ramas separadas para realizar cambios específicos en el código:

Primeramente, creamos ramas separadas, donde cada uno creó su propia rama desde el repositorio local:

1. Rama para diseño:  
`git checkout -b HP_v1_diseño`  
`git push origin HP_v1_diseño`
2. Rama para comentarios:  
`git checkout -b HB_v1_comentarios`  
`git push origin HB_v1_comentarios`

Esto permitió que cada uno de nosotros trabajara de manera independiente sin afectar la rama principal (main).

Para realizar cambios en nuestra respectiva rama realizamos las modificaciones necesarias, por ejemplo:

1. En la rama de diseño, se modificó el texto y la interfaz de la aplicación.
2. En la rama de comentarios, se añadieron comentarios detallados en el código para mejorar la documentación interna.

Posteriormente cada uno confirmó y subió sus cambios:

1. Cambios en diseño:  
`git add .`  
`git commit -m "Mejorando el diseño de la app"`

```
git push origin HP_v1_diseño
```

2. Cambios en comentarios:

```
git add .
```

```
git commit -m "Agregando comentarios al código"
```

```
git push origin HB_v1_comentarios
```

Esto permitió que cada rama estuviera actualizada en el repositorio remoto. Y una vez que ambos terminamos con nuestros cambios, fue necesario fusionar las modificaciones en la rama main. Para lo cual se realizó lo siguiente:

1. Primero se actualizó la rama main:

```
git checkout main
```

```
git pull origin main
```

2. Luego se fusionaron los cambios de la rama de comentarios:

```
git merge HB_v1_comentarios
```

```
git push origin main
```

3. Se fusionaron los cambios de la rama de diseño:

```
git merge HP_v1_diseño
```

```
git push origin main
```

Esto permitió que todos los cambios estuvieran sincronizados en la rama principal.

Si Git detectaba conflictos al hacer merge, los resolvímos manualmente editando los archivos afectados:

Un ejemplo de conflicto fue:

```
<<<<< HEAD  
(Código en main)  
=====  
(Código en HP_v1_diseño)  
>>>>> HP_v1_diseño
```

Se resolvió el conflicto seleccionando el código correcto y luego se confirmó el estado:

```
git add .
```

```
git commit -m "Resolviendo conflictos entre HP_v1_diseño y main"
```

```
git push origin main
```

Por otra parte, para recuperar versiones anteriores en caso de que fuera necesario volver a una versión anterior, se utilizaron los siguientes comandos:

1. Ver historial de cambios:  
`git log --oneline`
2. Volver temporalmente a una versión específica:  
`git checkout <id_commit>`
3. En caso de que quisieramos restaurar completamente una versión anterior:  
`git reset --hard <id_commit>`  
`git push --force origin main`

## **Lecciones de Conceptos básicos de Kotlin para Android y Aspectos básicos de Android con Compose**

Para la lección 1 de GreetingCard nosotros la teníamos ya hecha en el proyecto en AndroidStudio con el nombre de GreetingCard2, la cual subimos a github de la siguiente manera:

Desde el explorador de archivos, movimos la carpeta del proyecto GreetingCard2 desde la ruta original:

`C:\Users\Karla Cruz\AndroidStudioProjects\GreetingCard2`

A:

`C:\Users\Karla Cruz\AndroidStudioProjects\ProyectosCompartidos`

Esto para colocar el proyecto dentro de la carpeta que ya estaba vinculada a GitHub.

Posteriormente, abrimos Git Bash y navegamos a la carpeta del repositorio:

`cd "C:/Users/Karla Cruz/AndroidStudioProjects/ProyectosCompartidos"`

Esto permitió que Git reconociera el proyecto como parte del repositorio local. Y una vez dentro de la carpeta del repositorio, registramos los cambios:

1. Agregamos los archivos al área de preparación:  
`git add .`
2. Confirmamos los cambios con un mensaje descriptivo:  
`git commit -m " GreetingCard subida corregida"`

```
91806b7 GreetingCard subida corregida
```

3. Subimos los cambios al repositorio remoto en GitHub:  
`git push origin main`

Esto nos permitió que el proyecto apareciera reflejado en GitHub dentro de la carpeta GreetingCard2. Y, para confirmar que el proyecto efectivamente se subió, verificamos el estado con: `git status`

A lo cual Git nos mostró "your branch is up to date", lo cual nos indicaba que los cambios ya estaban sincronizados en GitHub. También teniendo abierto el navegador revisamos directamente el repositorio de GitHub para confirmar que el proyecto se reflejaba correctamente.

Una vez teniendo el proyecto de Android subido correctamente en nuestro repositorio lo siguiente que hicimos fue crear las ramas GC\_v1\_diseño y GC\_v1\_comentarios para separar los cambios.

Primero, navegamos a la carpeta del repositorio desde Git Bash:

```
cd "C:/Users/Karla  
Cruz/AndroidStudioProjects/ProyectosCompartidos/GreetingCard2"
```

Esto permitió que Git reconociera el repositorio y los archivos del proyecto.

Por otra parte, para crear la rama en la que uno de nosotros iba a trabajar en los comentarios, se ejecutó:

```
git checkout -b GC_v1_comentarios
```

Este comando creó una nueva rama llamada GC\_v1\_comentarios y se cambió automáticamente a esa rama.

Después de realizar los cambios en el código (en este caso agregar comentarios), se confirmaron los cambios con:

```
git add .  
git commit -m "Comentarios terminados en GC_v1"  
32d4a41 (origin/GC_v1_comentarios) Comentarios terminados en GC_v1
```

Luego se subió la rama a GitHub con:

```
git push origin GC_v1_comentarios
```

Con esto, la rama de comentarios quedó reflejada en GitHub.

Los mismos pasos se realizaron, pero para la rama de diseño GC\_v1\_diseño. Con esto, la rama de diseño también quedó reflejada en GitHub.

```
ab3b517 Modificando diseño de la app (Karla)
```

Una vez que los cambios estaban listos, realizamos la fusión en main para unir ambos trabajos:

1. Primero, nos colocamos en la rama main:  
`git checkout main`
2. Trajimos los cambios más recientes de GitHub para asegurarnos de que main estuviera actualizado:  
`git pull origin main`
3. Luego, fusionamos la rama de comentarios con main:  
`git merge GC_v1_comentarios`
4. Despues, subimos los cambios a GitHub:  
`git push origin main`
5. Y fusionamos la rama de diseño con main:  
`git merge GC_v1_diseño`
6. Finalmente, subimos el resultado final al repositorio remoto:  
`git push origin main`

En este punto, los cambios de diseño y comentarios quedaron reflejados en la rama principal (main).

```
1525d67 (GC_v1_diseño) Juntando diseño karla y comentarios yahir GC
```

Finalmente, si queremos ver alguna versión de las que hicimos en este proyecto ejecutamos en Git: `git log --oneline`



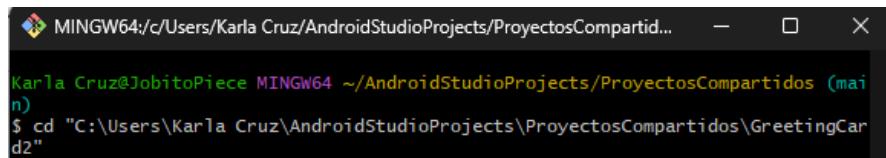
```
MINGW64:/c/Users/Karla Cruz/AndroidStudioProjects/ProyectosCompartidos
Karla Cruz@JobitoPiece MINGW64 /
$ cd "C:\Users\Karla Cruz\AndroidStudioProjects\ProyectosCompartidos"
Karla Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos (main)
$ git log --oneline
948f90 (HEAD -> main, origin/main, origin/HEAD, origin/AK_HMundo_diseño, AK_HMundo_diseño) Diseño de AK_HMundo_diseño (Main y Comandos)
39ef7db Merge branch 'main' of https://github.com/KarlaDenisseCruzSolis/ProyectosCompartidos
f07c28d Añadiendo AK_Cap1_HolaMundo configurado
e717cb5 lunch agregado
```

Esto nos mostrara nuestras distintas versiones, donde podemos observar que para el proyecto de GreetingCard2 le corresponden las siguientes:

Modificación realizada	Versión
Proyecto recién subido	91806b7
Diseño Karla sin comentarios de Yahir	ab3b517
Comentarios de Yahir	32d4a41
Proyecto actual de fusión del diseño Karla con el de Yahir	1525d67 (main)

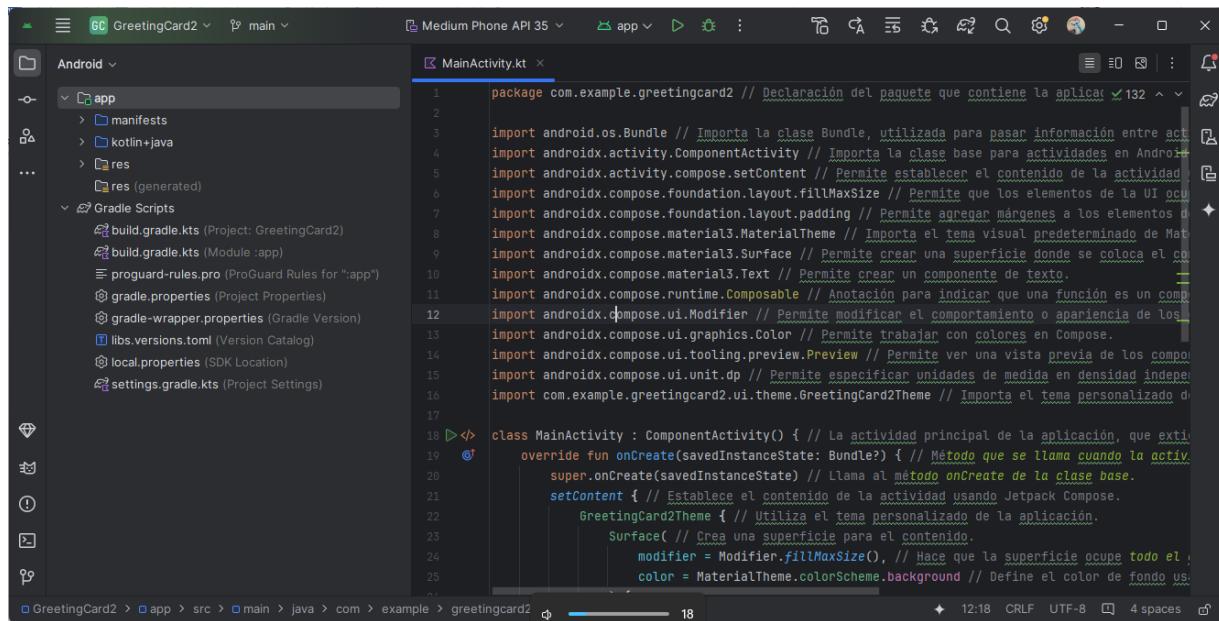
Para poder ver el historial de versiones del proyecto con sus identificadores únicos, como se mencionaba anteriormente se utiliza el comando: git log --oneline. Por ejemplo, si quisiéramos volver a alguna versión sería así:

Primero nos colocamos dentro del proyecto con el comando \$ cd "C:\Users\Karla Cruz\AndroidStudioProjects\ProyectosCompartidos\GreetingCard2":

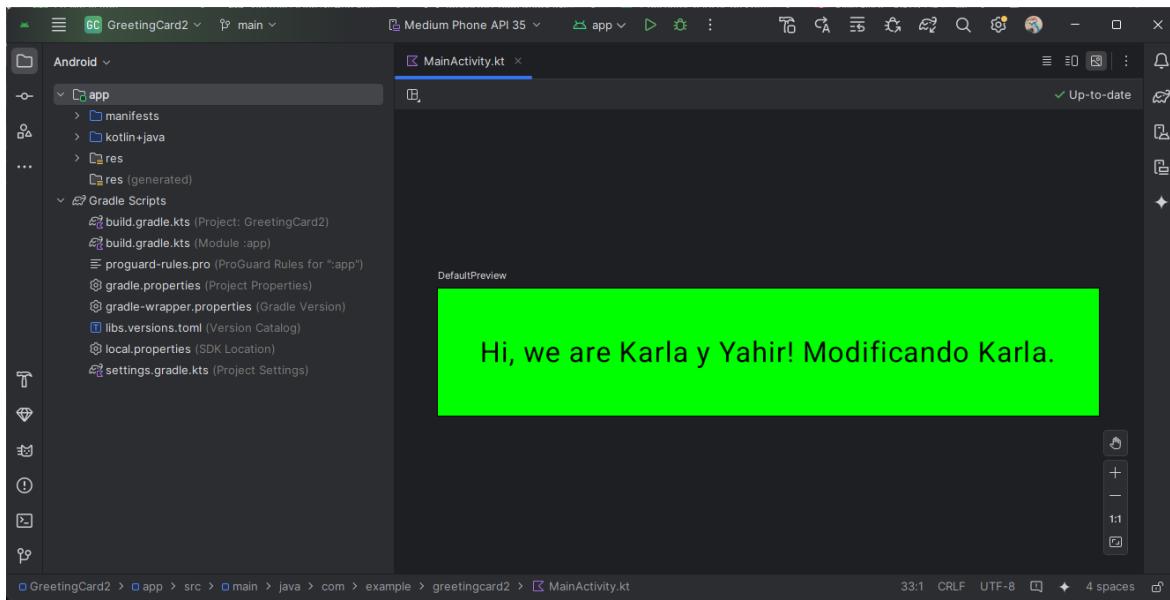


```
MINGW64:/c/Users/Karla Cruz/AndroidStudioProjects/ProyectosCompartid... - X
Karla Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos (main)
$ cd "C:\Users\Karla Cruz\AndroidStudioProjects\ProyectosCompartidos\GreetingCard2"
```

En este momento la última versión o la versión actual que tenemos de ese proyecto es la de la rama principal que es donde se fusionaron los cambios de diseño y comentarios (1525d67) (main), la cual luce de la siguiente manera:



```
>MainActivity.kt
1 package com.example.greetingcard2 // Declaración del paquete que contiene la aplicación
2
3 import android.os.Bundle // Importa la clase Bundle, utilizada para pasar información entre actividades
4 import androidx.activity.ComponentActivity // Importa la clase base para actividades en Android
5 import androidx.activity.compose.setContent // Permite establecer el contenido de la actividad
6 import androidx.compose.foundation.layout.fillMaxSize // Permite que los elementos de la UI ocupen toda la superficie
7 import androidx.compose.foundation.layout.padding // Permite agregar márgenes a los elementos de la UI
8 import androidx.compose.material3.MaterialTheme // Importa el tema visual predeterminado de Material
9 import androidx.compose.material3.Surface // Permite crear una superficie donde se coloca el contenido
10 import androidx.compose.material3.Text // Permite crear un componente de texto
11 import androidx.compose.runtime.Composable // Anotación para indicar que una función es un componente composable
12 import androidx.compose.ui.Modifier // Permite modificar el comportamiento o apariencia de los componentes
13 import androidx.compose.ui.graphics.Color // Permite trabajar con colores en Compose
14 import androidx.compose.ui.tooling.preview.Preview // Permite ver una vista previa de los componentes
15 import androidx.compose.ui.unit.dp // Permite especificar unidades de medida en densidad independiente
16 import com.example.greetingcard2.ui.theme.GreetingCard2Theme // Importa el tema personalizado de la aplicación
17
18 class MainActivity : ComponentActivity() { // La actividad principal de la aplicación, que extiende la clase ComponentActivity
19     override fun onCreate(savedInstanceState: Bundle?) { // Método que se llama cuando la actividad se crea
20         super.onCreate(savedInstanceState) // Llama al método onCreate de la clase base.
21         setContent { // Establece el contenido de la actividad usando Jetpack Compose.
22             GreetingCard2Theme { // Utiliza el tema personalizado de la aplicación.
23                 Surface( // Crea una superficie para el contenido.
24                     modifier = Modifier.fillMaxSize(), // Hace que la superficie ocupe todo el espacio disponible
25                     color = MaterialTheme.colorScheme.background // Define el color de fondo usado en el tema
26                 )
27             }
28         }
29     }
30 }
31
32 
```



Luego hacemos un `git checkout 91806b7` para poder ver la primera versión que habíamos subido:

```
MINGW64:/c/Users/Karla Cruz/AndroidStudioProjects/ProyectosCompartidos... - □ ×
Karla Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos/GreetingCard2 (main)
$ git checkout 91806b7
Updating files: 100% (7651/7651), done.
M      GreetingCard2/.idea/misc.xml
Note: switching to '91806b7'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

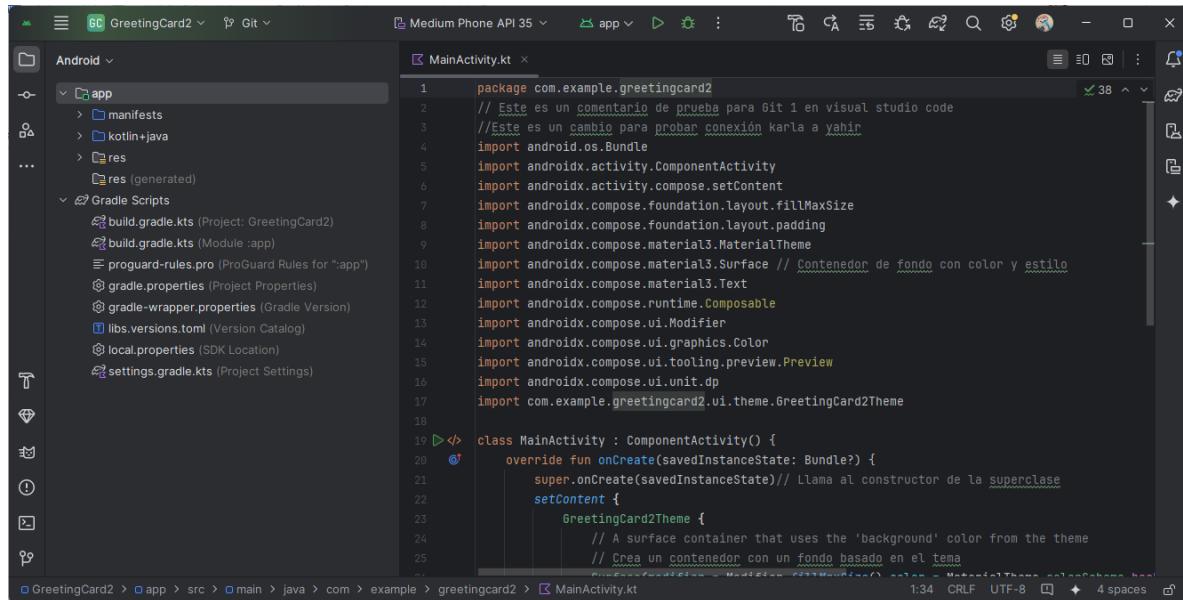
  git switch -c <new-branch-name>

Or undo this operation with:

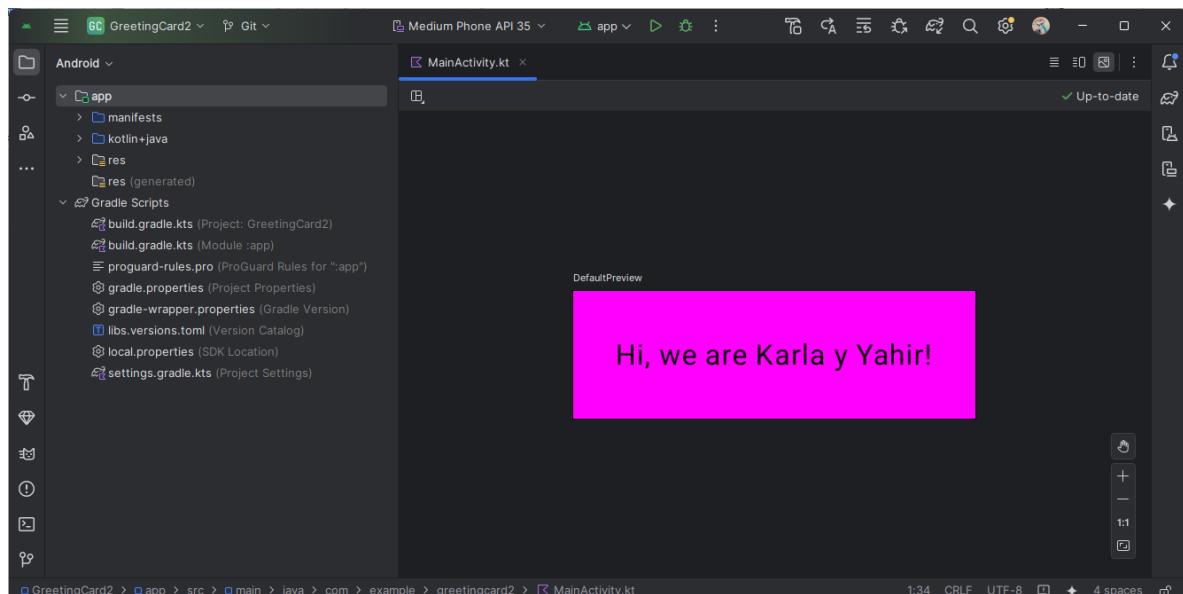
  git switch -

Turn off this advice by setting config variable advice.detachedHead to false
HEAD is now at 91806b7 GreetingCard subida corregida
```

Podemos ver cómo cambia:



The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right. The code editor displays the `MainActivity.kt` file. A specific line of code is highlighted with a green background and contains the text: `// Este es un comentario de prueba para Git 1 en visual studio code`. Below this, another line contains the text: `//Este es un cambio para probar conexión Karla a yahir`. The code itself is a standard Kotlin Activity implementation.



This screenshot shows the same Android Studio environment as the previous one, but the preview window in the bottom right corner displays a pink rectangular box with the text "Hi, we are Karla y Yahir!". This indicates that the application's UI has been updated to reflect the changes made in the code.

Si queremos cómo fue cuando sólo teníamos el diseño modificado hacemos un git checkout ab3b517:

```
Karla Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos/GreetingCard2 ((91806b7...))
$ git checkout ab3b517
M      GreetingCard2/.idea/misc.xml
Previous HEAD position was 91806b7 GreetingCard subida corregida
HEAD is now at ab3b517 Modificando diseño de la app (Karla)
```

Y observamos:

The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right. The code editor displays the `MainActivity.kt` file:

```
1 package com.example.greetingcard2
2 // Este es un comentario de prueba para Git 1 en visual studio code
3 //Este es un cambio para probar conexión karla a yahir
4
5 //Ejecuta el programa
6
7 //Mensaje de Karla a Yahir
8 import android.os.Bundle
9 import androidx.activity.ComponentActivity
10 import androidx.activity.compose.setContent
11 import androidx.compose.foundation.layout.fillMaxSize
12 import androidx.compose.foundation.layout.padding
13 import androidx.compose.material3.MaterialTheme
14 import androidx.compose.material3.Surface // Contenedor de fondo con color y estilo
15 import androidx.compose.material3.Text
16 import androidx.compose.runtime.Composable
17 import androidx.compose.ui.Modifier
18 import androidx.compose.ui.graphics.Color
19 import androidx.compose.ui.tooling.preview.Preview
20 import androidx.compose.ui.unit.dp
21 import com.example.greetingcard2.ui.theme.GreetingCard2Theme
22
23 <> class MainActivity : ComponentActivity() {
24     @Override fun onCreate(savedInstanceState: Bundle?) {
25         super.onCreate(savedInstanceState)// Llama al constructor de la superclase
26     }
27 }
```

The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right. The code editor displays the `MainActivity.kt` file. A preview window is open in the bottom right corner, showing the text "Hi, Karla y Yahir! Modificando Karla." in white on a green background.

Por otra parte, si queremos ver la versión donde teníamos los comentarios se ejecuta git checkout 32d4a41:

```

MINGW64:/c/Users/Karla Cruz/AndroidStudioProjects/ProyectosCompartidos... - □ ×
Karla Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos/Gree
tingCard2 (main)
$ git checkout 32d4a41
Updating files: 100% (7387/7387), done.
M     GreetingCard2/.idea/misc.xml
Note: switching to '32d4a41'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

    git switch -c <new-branch-name>

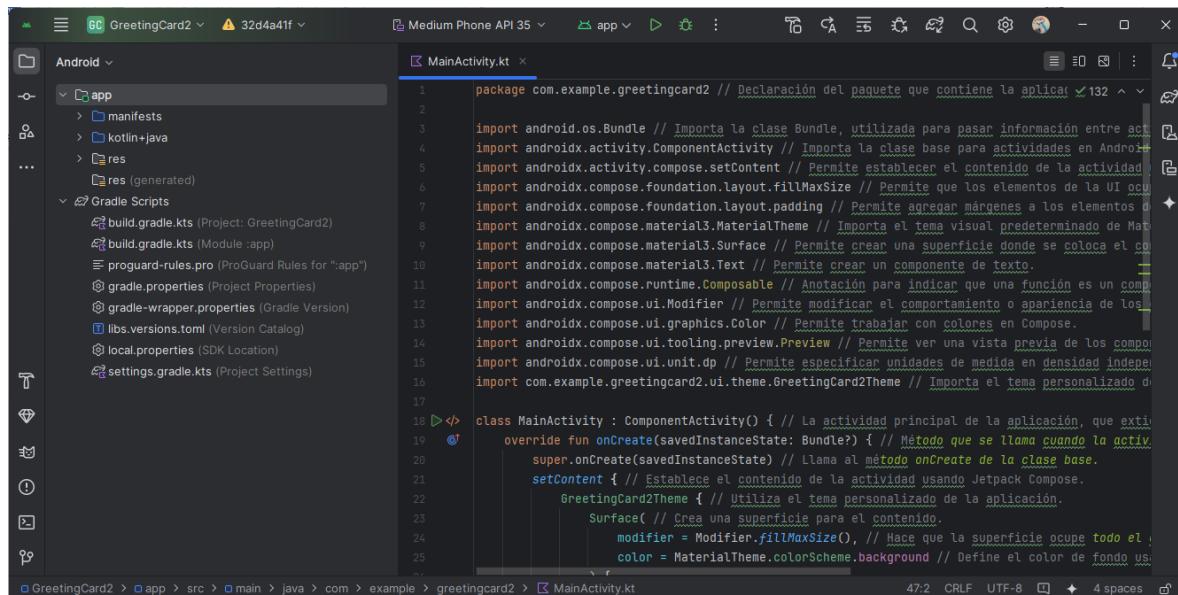
Or undo this operation with:

    git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 32d4a41 Comentarios terminados en GC_v1

```



Por último, si queremos volver a la última versión simplemente hacemos el git checkout main:

```

do so (now or later) by using -c with the switch command. Example:
    git switch -c <new-branch-name>
Or undo this operation with:
    git switch -
Turn off this advice by setting config variable advice.detachedHead to false
HEAD is now at 32d4a41 Comentarios terminados en GC_v1

Karla Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos/Gree
tingCard2 ((32d4a41...))
$ git checkout main
Updating files: 100% (7387/7387), done.
M     GreetingCard2/.idea/misc.xml
Previous HEAD position was 32d4a41 Comentarios terminados en GC_v1
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

Karla Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos/Gree
tingCard2 (main)
$ 

```

The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right. The code editor displays the `MainActivity.kt` file, which contains the following code:

```
1 package com.example.greetingcard2 // Declaración del paquete que contiene la aplicación
2
3 import android.os.Bundle // Importa la clase Bundle, utilizada para pasar información entre actividades
4 import androidx.activity.ComponentActivity // Importa la clase base para actividades en Android
5 import androidx.activity.compose.setContent // Permite establecer el contenido de la actividad
6 import androidx.compose.foundation.layout.fillMaxSize // Permite que los elementos de la UI ocupen todo el espacio disponible
7 import androidx.compose.foundation.layout.padding // Permite agregar márgenes a los elementos de la UI
8 import androidx.compose.material3.MaterialTheme // Importa el tema visual predeterminado de Material
9 import androidx.compose.material3.Surface // Permite crear una superficie donde se coloca el contenido
10 import androidx.compose.material3.Text // Permite crear un componente de texto
11 import androidx.compose.runtime.Composable // Anotación para indicar que una función es un componente
12 import androidx.compose.ui.Modifier // Permite modificar el comportamiento o apariencia de los componentes
13 import androidx.compose.ui.graphics.Color // Permite trabajar con colores en Compose
14 import androidx.compose.ui.tooling.preview.Preview // Permite ver una vista previa de los componentes
15 import androidx.compose.ui.unit.dp // Permite especificar unidades de medida independientes de la densidad
16 import com.example.greetingcard2.ui.theme.GreetingCard2Theme // Importa el tema personalizado de la aplicación
17
18 class MainActivity : ComponentActivity() { // La actividad principal de la aplicación, que extiende la clase ComponentActivity
19     override fun onCreate(savedInstanceState: Bundle?) { // Método que se llama cuando la actividad se crea
20         super.onCreate(savedInstanceState) // Llama al método onCreate de la clase base
21         setContent { // Establece el contenido de la actividad usando Jetpack Compose
22             GreetingCard2Theme { // Utiliza el tema personalizado de la aplicación
23                 Surface( // Crea una superficie para el contenido
24                     modifier = Modifier.fillMaxSize(), // Hace que la superficie ocupe todo el espacio disponible
25                     color = MaterialTheme.colorScheme.background // Define el color de fondo usando el tema
                }
            }
        }
    }
}
```

The screenshot shows the Android Studio interface with the project structure on the left and the preview window on the right. The preview window displays the application's UI with the text "Hi, we are Karla y Yahir! Modificando Karla.". A tooltip labeled "DefaultPreview" is visible above the preview window. A small window titled "Herramienta Recortes" (Clipboard Tool) is open in the bottom right corner, showing a screenshot of the preview window.

Y así con cada uno de los proyectos que ya tenemos hechos.

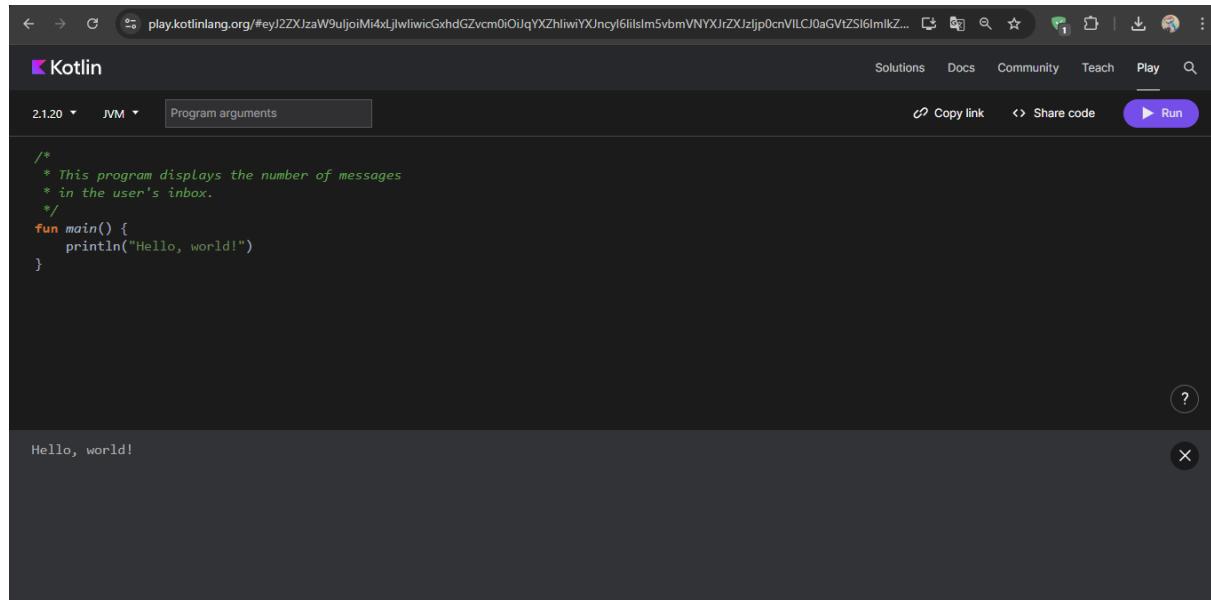
En cuanto al orden correcto de las lecciones o unidades este se muestra a continuación:

## Unidad 1: Tu primera app para Android

### Ruta de Aprendizaje 1 (Introducción a Kotlin)

#### Tu primer programa en Kotlin

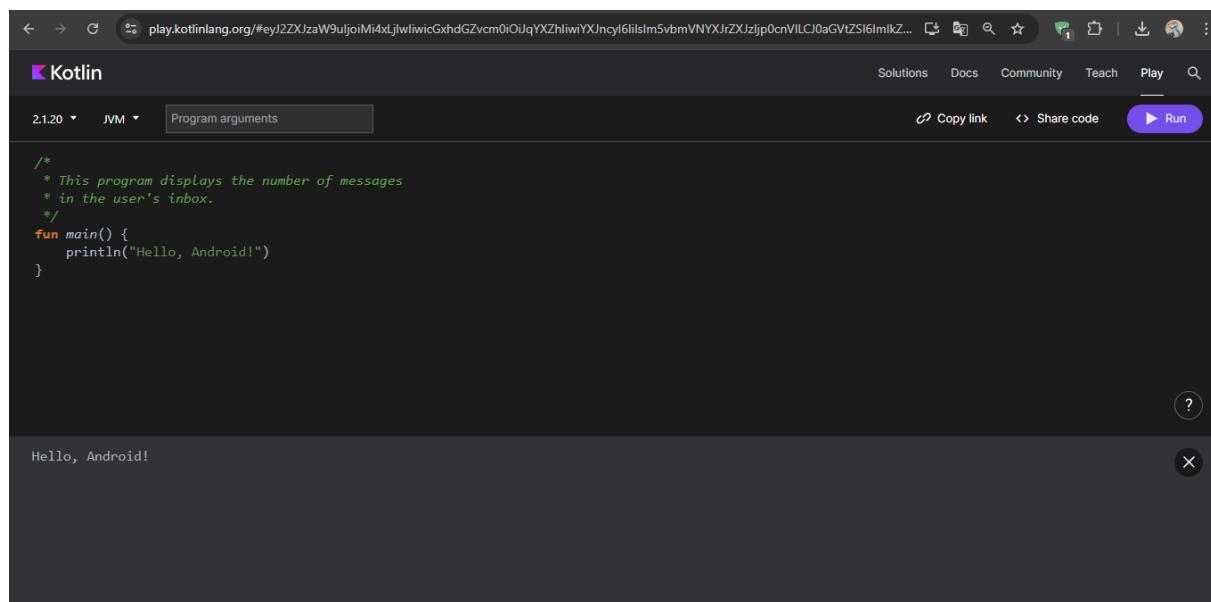
En este codelab se explica la función principal `fun main()` y cómo usar `println()` para imprimir texto. Se enseña cómo modificar el mensaje, por ejemplo, cambiando "Hello, world!" por "Hello, Android!", y cómo repetir líneas para imprimir el mensaje más de una vez.



The screenshot shows the Kotlin playground interface. The code editor contains the following Kotlin code:

```
/*
 * This program displays the number of messages
 * in the user's inbox.
 */
fun main() {
    println("Hello, world!")
}
```

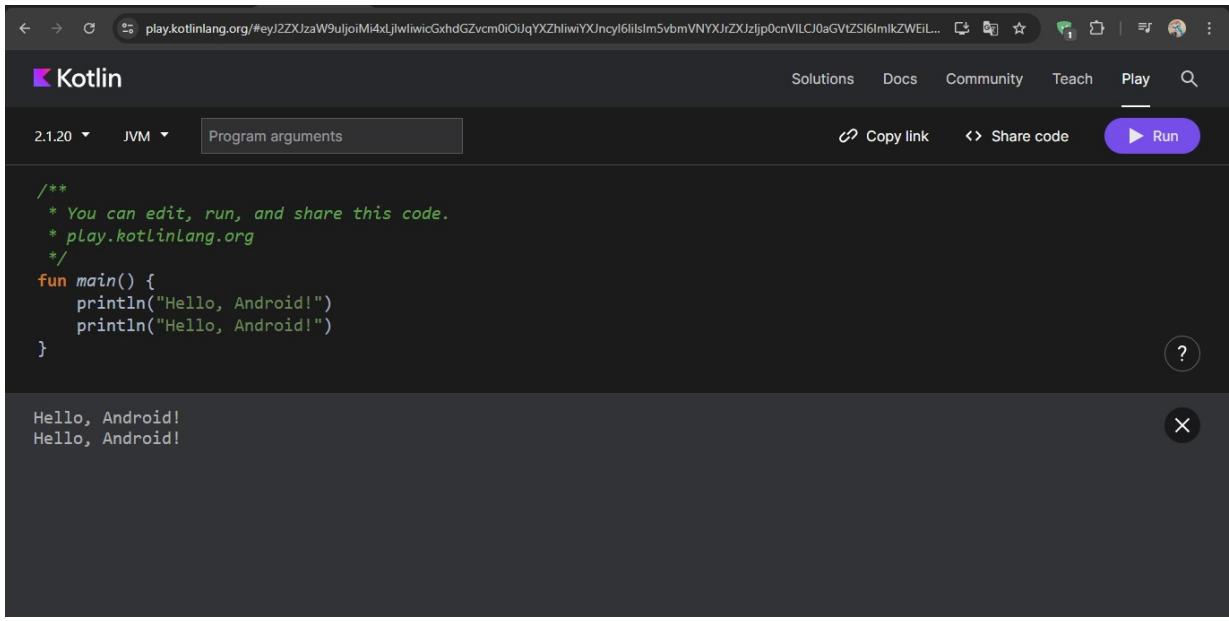
The output window below the code editor shows the result of running the program: "Hello, world!".



The screenshot shows the Kotlin playground interface. The code editor contains the following Kotlin code:

```
/*
 * This program displays the number of messages
 * in the user's inbox.
 */
fun main() {
    println("Hello, Android!")
}
```

The output window below the code editor shows the result of running the program: "Hello, Android!".



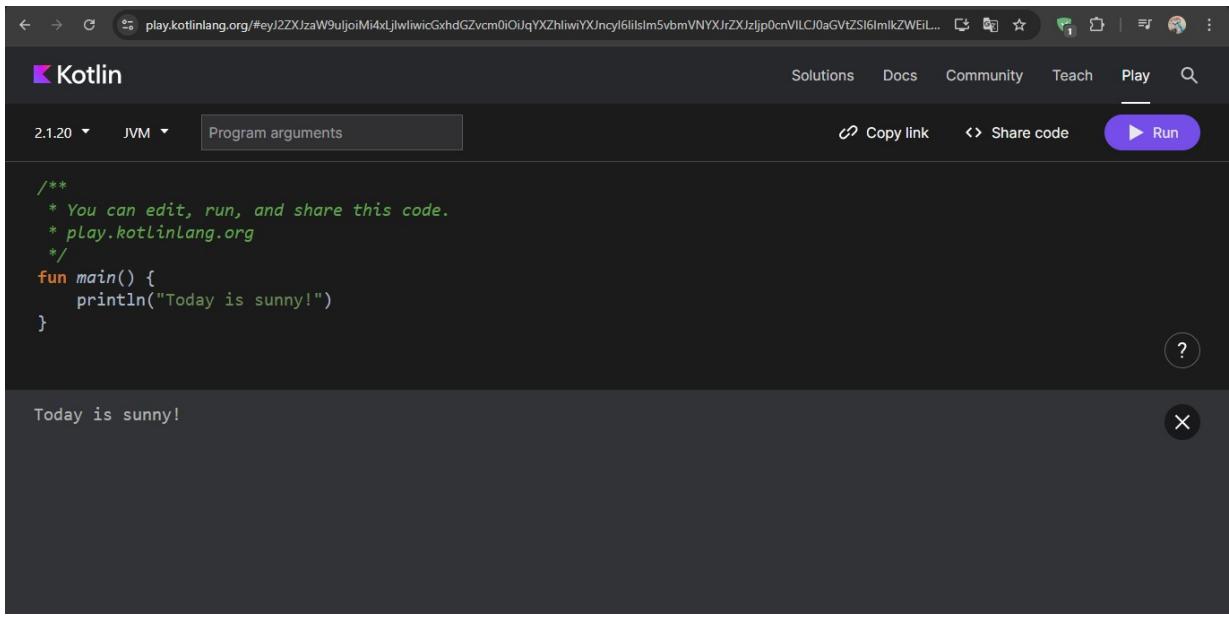
The screenshot shows a web-based Kotlin code editor at [play.kotlinlang.org](https://play.kotlinlang.org/). The code in the editor is:

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    println("Hello, Android!")
    println("Hello, Android!")
}
```

The output window below the editor shows the execution results:

```
Hello, Android!
Hello, Android!
```

También se plantean ejercicios prácticos como: adivinar la salida de fragmentos de código, reorganizar líneas para obtener un orden específico (como días de la semana), y corregir errores comunes como falta de paréntesis, nombres de funciones mal escritos (como `printLine` en lugar de `println`), o el uso incorrecto de llaves y paréntesis en la estructura del programa.



The screenshot shows a web-based Kotlin code editor at [play.kotlinlang.org](https://play.kotlinlang.org/). The code in the editor is:

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    println("Today is sunny!")
}
```

The output window below the editor shows the execution results:

```
Today is sunny!
```

Kotlin

Solutions Docs Community Teach Play

2.1.20 JVM Program arguments

Copy link Share code Run

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    println("1")
    println("2")
    println("3")
}
```

1  
2  
3

Kotlin

Solutions Docs Community Teach Play

2.1.20 JVM Program arguments

Copy link Share code Run

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    println("I'm")
    println("learning")
    println("Kotlin!")
}
```

I'm  
learning  
Kotlin!

Kotlin

Solutions Docs Community Teach Play

2.1.20 ▾ JVM ▾ Program arguments

Copy link Share code Run

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    println("Tuesday")
    println("Thursday")
    println("Wednesday")
    println("Friday")
    println("Monday")
}
```

Tuesday  
Thursday  
Wednesday  
Friday  
Monday

Kotlin

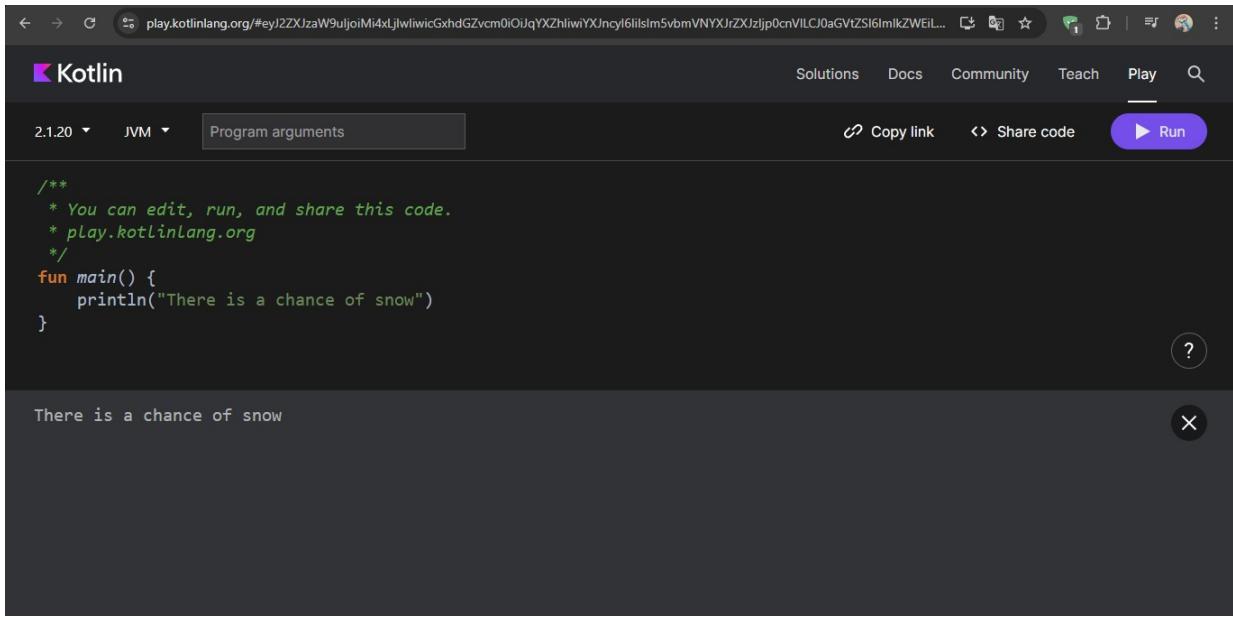
Solutions Docs Community Teach Play

2.1.20 ▾ JVM ▾ Program arguments

Copy link Share code Run

```
/**
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    println("Tomorrow is rainy")
}
```

Tomorrow is rainy



Kotlin

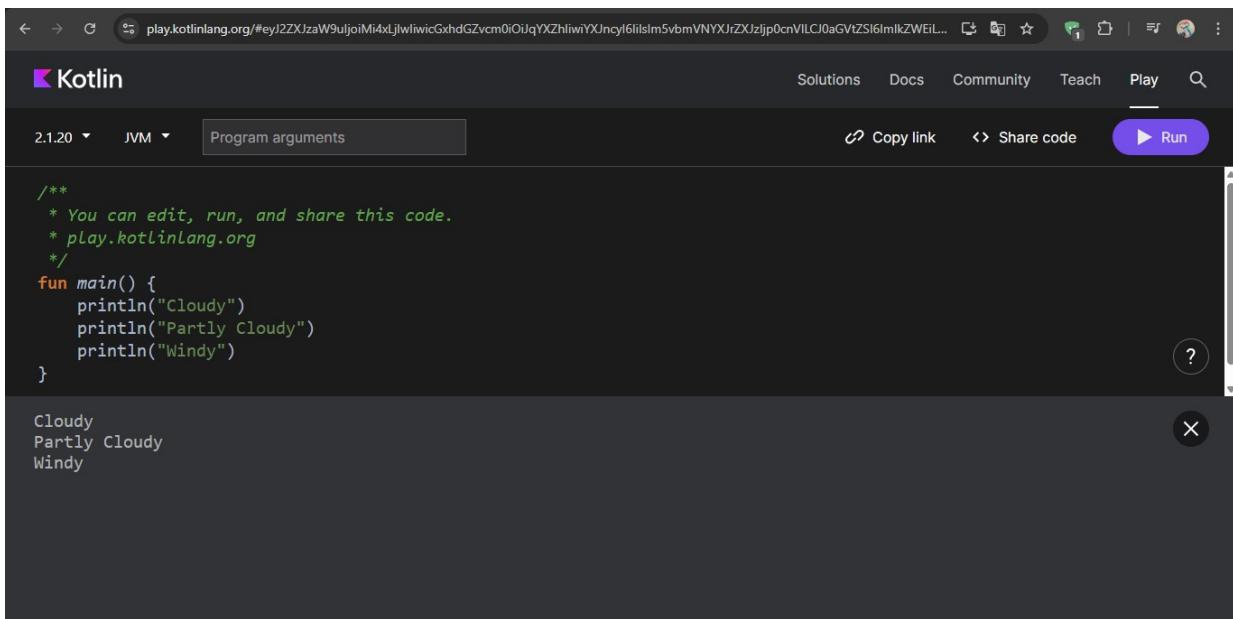
Solutions Docs Community Teach Play

2.1.20 JVM Program arguments

Copy link Share code Run

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    println("There is a chance of snow")
}
```

There is a chance of snow



Kotlin

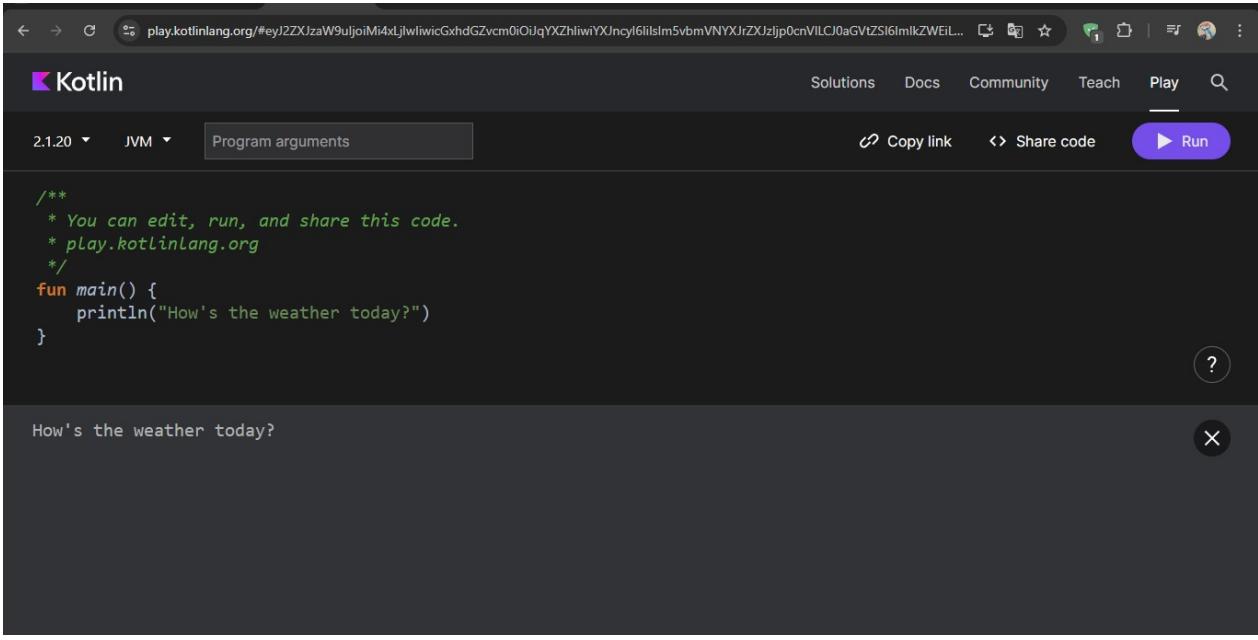
Solutions Docs Community Teach Play

2.1.20 JVM Program arguments

Copy link Share code Run

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    println("Cloudy")
    println("Partly Cloudy")
    println("Windy")
}
```

Cloudy  
Partly Cloudy  
Windy



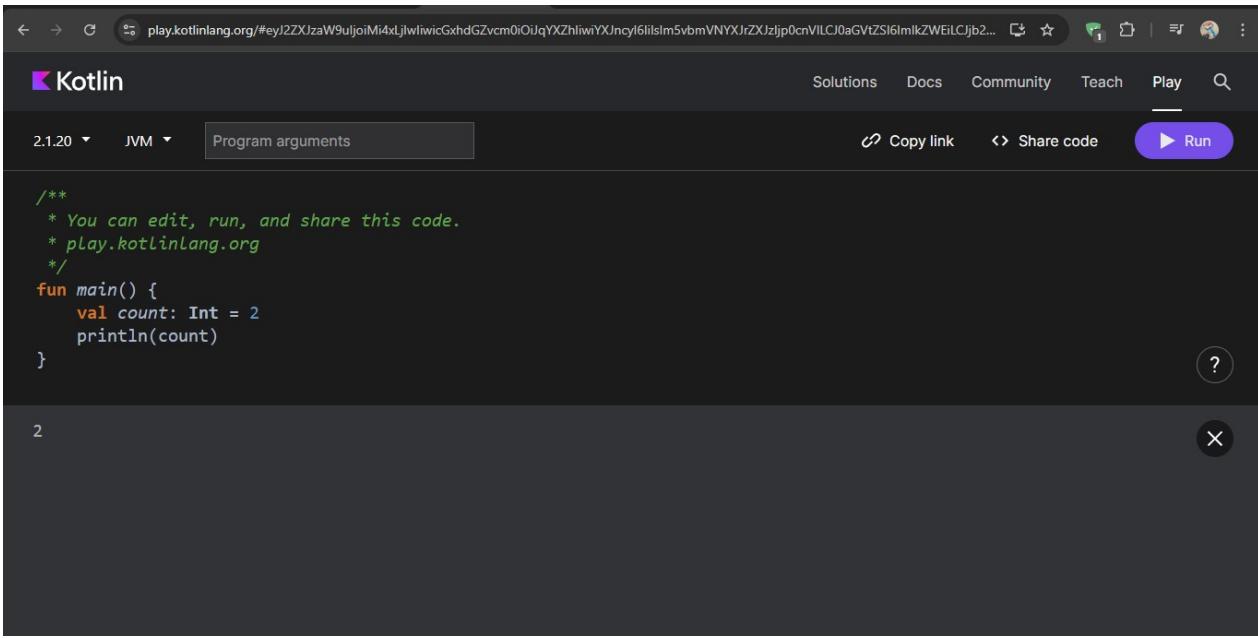
The screenshot shows the Kotlin Play IDE interface. At the top, there are navigation icons, a URL bar with the address `play.kotlinlang.org/#eyJ2ZXJzaW9uJoiMi4xLjwlwicGxhdGZvcm0iOiJqYXZhiwiYXJncyl6iislm5vbmVNYXIrZXJzlp0cnVILC0aGVtZSI6ImlkZWEl...`, and a search bar. Below the header, the title "Kotlin" is displayed, followed by version "2.1.20" and "JVM". A "Program arguments" input field is present. On the right side of the header, there are links for "Solutions", "Docs", "Community", "Teach", "Play", and a search icon. Below the header, the code editor contains the following Kotlin code:

```
/**  
 * You can edit, run, and share this code.  
 * play.kotlinLang.org  
 */  
fun main() {  
    println("How's the weather today?")  
}
```

On the right side of the code editor, there are "Copy link", "Share code", and "Run" buttons. Below the code editor, the output window displays the result of the execution: "How's the weather today?".

## Crea y usa variables en Kotlin

Se explica cómo declarar y utilizar variables, tanto inmutables (val) como mutables (var). Las variables inmutables no pueden cambiar su valor después de ser asignadas, mientras que las mutables sí pueden modificarse durante la ejecución del programa.



The screenshot shows the Kotlin Play IDE interface. At the top, there are navigation icons, a URL bar with the address `play.kotlinlang.org/#eyJ2ZXJzaW9uJoiMi4xLjwlwicGxhdGZvcm0iOiJqYXZhiwiYXJncyl6iislm5vbmVNYXIrZXJzlp0cnVILC0aGVtZSI6ImlkZWEl...`, and a search bar. Below the header, the title "Kotlin" is displayed, followed by version "2.1.20" and "JVM". A "Program arguments" input field is present. On the right side of the header, there are links for "Solutions", "Docs", "Community", "Teach", "Play", and a search icon. Below the header, the code editor contains the following Kotlin code:

```
/**  
 * You can edit, run, and share this code.  
 * play.kotlinLang.org  
 */  
fun main() {  
    val count: Int = 2  
    println(count)  
}
```

On the right side of the code editor, there are "Copy link", "Share code", and "Run" buttons. Below the code editor, the output window displays the result of the execution: "2".

Kotlin

Solutions Docs Community Teach Play

2.1.20 JVM Program arguments

```
/*
 * Unidad 1
 */

fun main() {
    val count: Int = 2
    println("You have count unread messages.")
}
```

You have count unread messages.

Kotlin

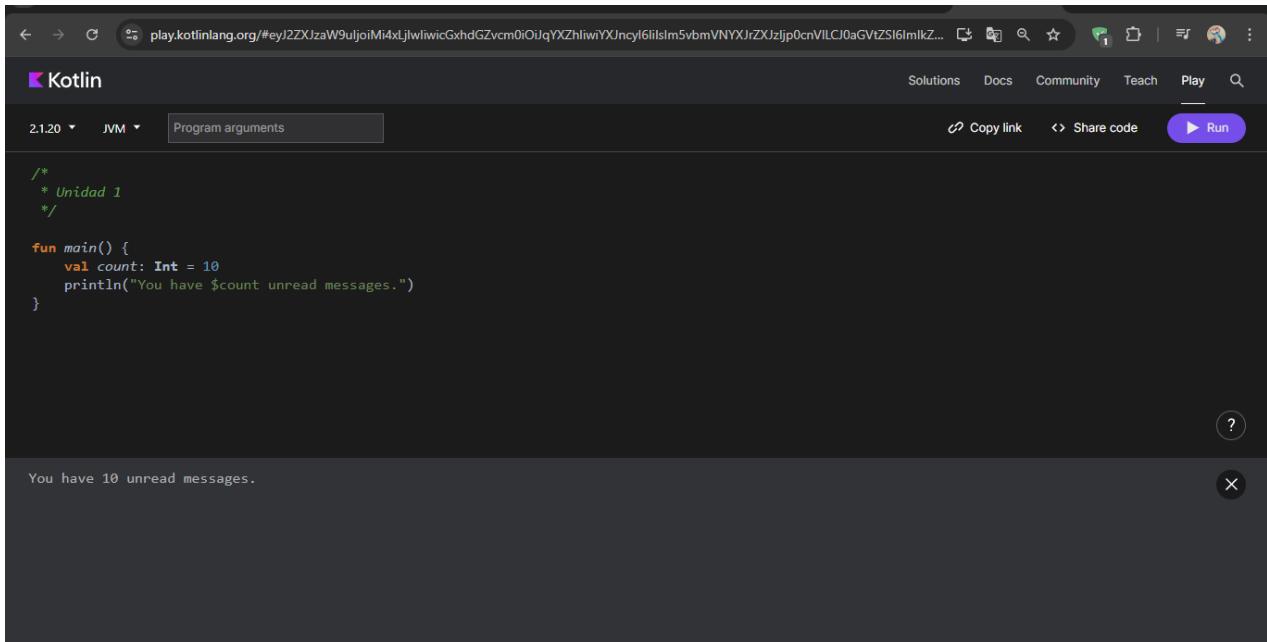
Solutions Docs Community Teach Play

2.1.20 JVM Program arguments

```
/*
 * Unidad 1
 */

fun main() {
    val count: Int = 2
    println("You have $count unread messages.")
}
```

You have 2 unread messages.



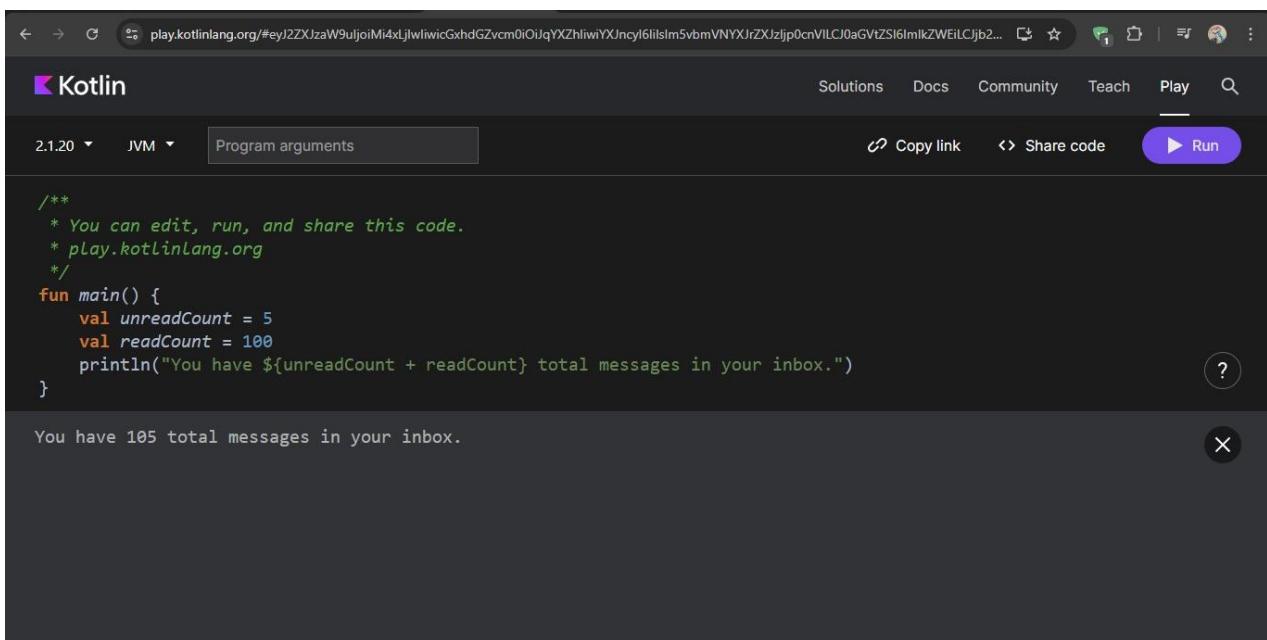
The screenshot shows a web-based Kotlin playground interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the bar, the version is set to 2.1.20 and the JVM target is selected. A "Program arguments" input field is present. On the right, there are buttons for "Copy link", "Share code", and "Run". The main area contains the following Kotlin code:

```
/*
 * Unidad 1
 */

fun main() {
    val count: Int = 10
    println("You have $count unread messages.")
}
```

When the code is run, the output "You have 10 unread messages." is displayed below the code editor.

Además, se nos muestra que en Kotlin, el símbolo “\$” se utiliza dentro de cadenas para insertar el valor de una variable, y “\${...}” permite incluir expresiones o cálculos directamente en el texto. Esto facilita combinar texto con datos sin tener que concatenar manualmente.



The screenshot shows another instance of the Kotlin playground. The setup is identical to the first one. The code is as follows:

```
/**
 * You can edit, run, and share this code.
 * play.kotlinLang.org
 */
fun main() {
    val unreadCount = 5
    val readCount = 100
    println("You have ${unreadCount + readCount} total messages in your inbox.")
}
```

The output "You have 105 total messages in your inbox." is shown at the bottom of the screen.

Kotlin

Solutions Docs Community Teach Play

2.1.20 JVM Program arguments

```
* You can edit, run, and share this code.
* play.kotlinlang.org
*/
fun main() {
    val numberOfPhotos = 100
    val photosDeleted = 10
    println("$numberOfPhotos photos")
    println("$photosDeleted photos deleted")
    println("${numberOfPhotos - photosDeleted} photos left")
}
```

100 photos  
10 photos deleted  
90 photos left

Kotlin

Solutions Docs Community Teach Play

2.1.20 JVM Program arguments

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    var cartTotal = 0
    cartTotal = 20
    println("Total: $cartTotal")
}
```

Total: 20

Kotlin

Solutions Docs Community Teach Play

2.1.20 JVM Program arguments

```
* You can edit, run, and share this code.
* play.kotlinLang.org
*/
fun main() {
    var cartTotal = 0
    println("Total: $cartTotal")

    cartTotal = 20
    println("Total: $cartTotal")
}
```

Total: 0  
Total: 20

Kotlin

Solutions Docs Community Teach Play

2.1.20 JVM Program arguments

```
/*
 * You can edit, run, and share this code.
 * play.kotlinLang.org
 */
fun main() {
    val count: Int = 10
    println("You have $count unread messages.")
}
```

You have 10 unread messages.

The screenshot shows the Kotlin Play IDE interface. At the top, there are navigation icons, a search bar, and a URL: play.kotlinlang.org/#eyI2ZXJzaW9uljoiMi4xLjlwliwicGxhdGZvcn0iOiJqYXZhiwiYXJncyl6lislm5vbmVNYXJrZXJzlp0cnVILCJ0aGVtZSI6ImlkZWEiLCJjb2... . Below the URL is the Kotlin logo and the word "Kotlin". The version "2.1.20" and "JVM" are selected. A "Program arguments" input field is empty. On the right, there are buttons for "Copy link", "Share code", and a purple "Run" button. The main code editor contains the following code:

```
/**  
 * You can edit, run, and share this code.  
 * play.kotlinlang.org  
 */  
fun main() {  
    var count: Int = 10  
    println("You have $count unread messages.")  
}
```

The output window below the editor shows the result of running the code: "You have 10 unread messages." with a close button (X) to its right.

También se nos muestra cómo modificar el valor de una variable utilizando operaciones de incremento (“count++” o “count = count + 1”) y decremento (“count—”). Estas operaciones ajustan el valor de la variable en una unidad: “count++” aumenta el valor de “count” en 1, mientras que “count—“ lo disminuye en 1.

The screenshot shows the Kotlin Play IDE interface. At the top, there are navigation icons, a search bar, and a URL: play.kotlinlang.org/#eyI2ZXJzaW9uljoiMi4xLjlwliwicGxhdGZvcn0iOiJqYXZhiwiYXJncyl6lislm5vbmVNYXJrZXJzlp0cnVILCJ0aGVtZSI6ImlkZWEiLCJjb2... . Below the URL is the Kotlin logo and the word "Kotlin". The version "2.1.20" and "JVM" are selected. A "Program arguments" input field is empty. On the right, there are buttons for "Copy link", "Share code", and a purple "Run" button. The main code editor contains the following code:

```
/*  
 * Unidad 1  
 */  
fun main() {  
    var count = 10  
    println("You have $count unread messages.")  
    count = count + 1  
    println("You have $count unread messages.")  
}
```

The output window below the editor shows the result of running the code: "You have 10 unread messages." followed by "You have 11 unread messages." with a close button (X) to its right.

Kotlin

Solutions Docs Community Teach Play

2.1.20 JVM Program arguments

Copy link Share code Run

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    var count = 10
    println("You have $count unread messages.")
    count++
    println("You have $count unread messages.")
}
```

You have 10 unread messages.  
You have 11 unread messages.

Kotlin

Solutions Docs Community Teach Play

2.1.20 JVM Program arguments

Copy link Share code Run

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    var count = 10
    println("You have $count unread messages.")
    count--
    println("You have $count unread messages.")
}
```

You have 10 unread messages.  
You have 9 unread messages.

Kotlin

Solutions Docs Community Teach Play

2.1.20 JVM Program arguments

```
/*
 * This program displays the number of messages
 * in the user's inbox.
 */
fun main() {
    // Create a variable for the number of unread messages.
    var count = 10
    println("You have $count unread messages.")

    // Decrease the number of messages by 1.
    count--
    println("You have $count unread messages.")
}
```

You have 10 unread messages.  
You have 9 unread messages.

Kotlin

Solutions Docs Community Teach Play

2.1.20 JVM Program arguments

```
/*
 * Unidad 1
 */
fun main() {
    val trip1: Double = 3.20
    val trip2: Double = 4.10
    val trip3: Double = 1.72
    val totalTripLength: Double = trip1 + trip2 + trip3
    println("$totalTripLength miles left to destination")
}
```

9.02 miles left to destination

The screenshot shows the Kotlin Play IDE interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the navigation bar, the code editor displays the following Kotlin code:

```
/*
 * Unidad 1
 */

fun main() {
    val nextMeeting = "Next meeting:"
    val date = "January 1"
    val reminder = nextMeeting + date
    println(reminder)
}
```

The output window at the bottom shows the result of running the code: "Next meeting:January 1".

This screenshot shows the same setup as the first one, but with a modified code snippet. The code now includes an additional string concatenation step:

```
/*
 * Unidad 1
 */

fun main() {
    val nextMeeting = "Next meeting: "
    val date = "January 1"
    val reminder = nextMeeting + date + " at work"
    println(reminder)
}
```

The output window shows the updated result: "Next meeting: January 1 at work".

También se abordan distintos tipos de datos como numéricos, textos y booleanos, y se muestra cómo se pueden asignar y usar en el código, así como la importancia de los comentarios, tanto de una sola línea (//) como de varias líneas (\* \*), para mejorar la legibilidad del código y facilitar su comprensión. Además, se destaca la buena práctica de utilizar nombres descriptivos para las variables y agregar comentarios que expliquen su propósito, especialmente cuando el código es complejo. Finalmente, se muestra cómo las variables pueden ser utilizadas en el flujo del programa, como en cálculos o para mostrar resultados en pantalla.

The screenshot shows the Kotlin Play IDE interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, and Play. Below the navigation bar, the version is set to 2.1.20 and the JVM target is selected. A "Program arguments" input field contains the string "Say hello". On the right side of the editor, there are buttons for Copy link, Share code, and Run. The main code area contains a single-line comment and a main function:

```
/*
 * Unidad 1
 */
fun main() {
    println("Say hello")
}
```

The output window below the code shows the result of the execution: "Say hello".

This screenshot shows the same or a very similar setup in the Kotlin Play IDE. The code is identical to the one above, but the output window shows the result of the execution: "true".

```
/*
 * Unidad 1
 */
fun main() {
    val notificationsEnabled: Boolean = true
    println(notificationsEnabled)
}
```

The screenshot shows the Kotlin Play IDE interface. The code editor contains the following Kotlin code:

```
/*
 * Unidad 1
 */

fun main() {
    val notificationsEnabled: Boolean = false
    println(notificationsEnabled)
}
```

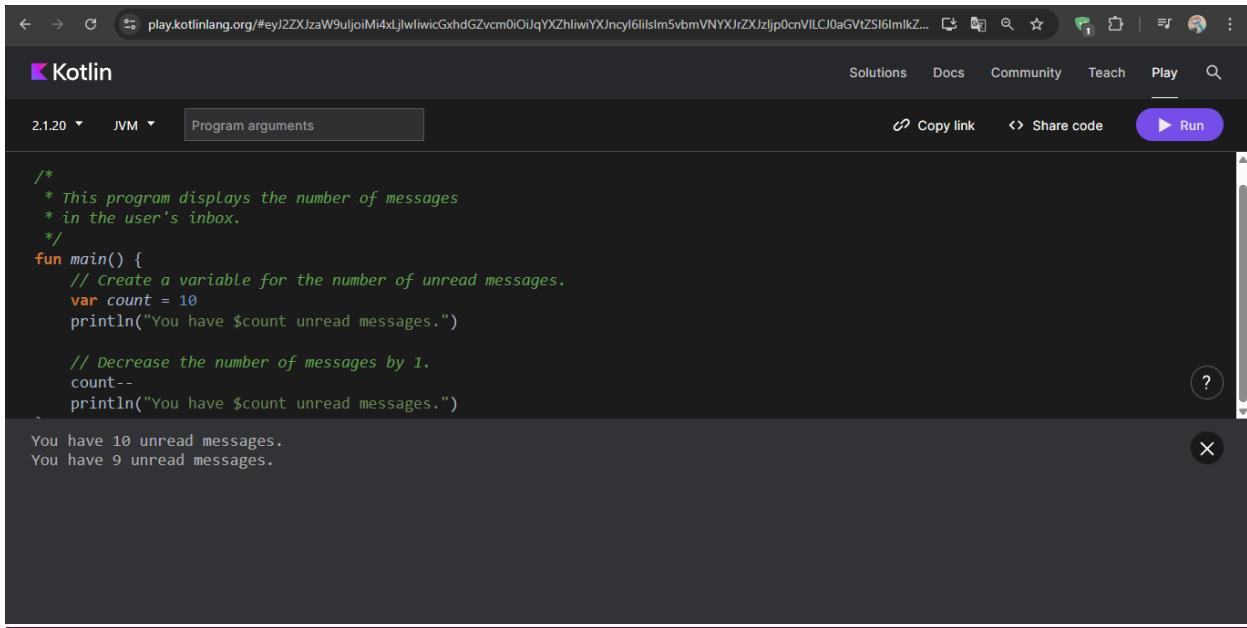
The output window below the code editor displays the result of the program's execution: "false".

The screenshot shows the Kotlin Play IDE interface. The code editor contains the following modified Kotlin code:

```
/*
 * Unidad 1
 */

fun main() {
    val notificationsEnabled: Boolean = false
    println("Are notifications enabled? " + notificationsEnabled)
}
```

The output window below the code editor displays the result of the program's execution: "Are notifications enabled? false".



The screenshot shows the Kotlin Play IDE interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the navigation bar, the version is listed as 2.1.20 and the JVM target is selected. A "Program arguments" input field is present. On the right side, there are buttons for "Copy link", "Share code", and "Run". The main area displays the following Kotlin code:

```
/*
 * This program displays the number of messages
 * in the user's inbox.
 */
fun main() {
    // Create a variable for the number of unread messages.
    var count = 10
    println("You have $count unread messages.")

    // Decrease the number of messages by 1.
    count--
    println("You have $count unread messages.")
}
```

Below the code, the output window shows two lines of text: "You have 10 unread messages." and "You have 9 unread messages.", indicating the execution of the program.

## Cómo crear y usar funciones en kotlin

En Kotlin, una función se define utilizando la palabra clave `fun`, y permite agrupar fragmentos de código reutilizable, lo que facilita el mantenimiento de programas grandes y evita la repetición innecesaria. Las funciones pueden recibir parámetros, que son variables definidas entre paréntesis y separadas por comas, y pueden devolver valores que pueden almacenarse para su uso posterior. Al llamar una función, se deben pasar argumentos que coincidan con los tipos y el orden de los parámetros, aunque Kotlin también permite el uso de argumentos con nombre, lo cual facilita la lectura del código y permite cambiar el orden en que se pasan los valores sin alterar el resultado.

The screenshot shows the Kotlin Play IDE interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the navigation bar, the version is set to 2.1.20 and the JVM target is selected. A "Program arguments" input field contains no text. On the right side of the toolbar are "Copy link", "Share code", and a purple "Run" button. The main workspace displays the following Kotlin code:

```
fun main() {
    birthdayGreeting()
}

fun birthdayGreeting() {
    println("Happy Birthday, Rover!")
    println("You are now 5 years old!")
}
```

Below the code, the output window shows the results of running the program:

```
Happy Birthday, Rover!
You are now 5 years old!
```

Ilustración 1 Define una función y llámala

The screenshot shows the Kotlin Play IDE interface. The setup is identical to Illustration 1: version 2.1.20, JVM target, and no program arguments. The "Run" button is purple. The workspace contains the following Kotlin code:

```
fun main() {
    println(birthdayGreeting())
}

fun birthdayGreeting(): String {
    val nameGreeting = "Happy Birthday, Rover!"
    val ageGreeting = "You are now 5 years old!"
    return "$nameGreeting\n$ageGreeting"
}
```

The output window shows the results of running the program:

```
Happy Birthday, Rover!
You are now 5 years old!
```

Ilustración 2 Muestra un valor de una función

The screenshot shows the Kotlin Play IDE interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the navigation bar, the version is set to 2.1.20 and the JVM is selected. A "Program arguments" input field is present. On the right, there are buttons for "Copy link", "Share code", and a purple "Run" button. The main area displays the following Kotlin code:

```
fun main() {
    println(birthdayGreeting("Rover"))
    println(birthdayGreeting("Rex"))
}

fun birthdayGreeting(name: String): String {
    val nameGreeting = "Happy Birthday, $name!"
    val ageGreeting = "You are now 5 years old!"
    return "$nameGreeting\n$ageGreeting"
}
```

Below the code, the output window shows the results of running the program:

```
Happy Birthday, Rover!
You are now 5 years old!
Happy Birthday, Rover!
You are now 5 years old!
```

Ilustración 3 Agrega un parámetro a la función birthdayGreeting()

The screenshot shows the Kotlin Play IDE interface. The setup is identical to Illustration 3. The main area displays the following updated Kotlin code:

```
fun main() {
    println(birthdayGreeting("Rover", 5))
    println(birthdayGreeting("Rex", 2))
}

fun birthdayGreeting(name: String, age: Int): String {
    val nameGreeting = "Happy Birthday, $name!"
    val ageGreeting = "You are now $age years old!"
    return "$nameGreeting\n$ageGreeting"
}
```

Below the code, the output window shows the results of running the program:

```
Happy Birthday, Rover!
You are now 5 years old!
Happy Birthday, Rover!
You are now 5 years old!
```

Ilustración 4 Funciones con varios parámetros

The screenshot shows the Kotlin Play codelab interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the navigation bar, there are dropdown menus for '2.1.20' and 'JVM'. A 'Program arguments' input field contains the values 'Rover' and '2'. To the right of the input field are 'Copy link', 'Share code', and a purple 'Run' button. The main area displays the following Kotlin code:

```
fun main() {
    println(birthdayGreeting(name = "Rover", age = 5))
    println(birthdayGreeting(age = 2, name = "Rex"))
}

fun birthdayGreeting(name: String, age: Int): String {
    val nameGreeting = "Happy Birthday, $name!"
    val ageGreeting = "You are now $age years old!"
    return "$nameGreeting\n$ageGreeting"
}
```

Below the code, the output is shown in a dark box:

```
Happy Birthday, Rover!
You are now 5 years old!
Happy Birthday, Rex!
You are now 2 years old!
```

Ilustración 5 Argumentos con nombre

Además, en este codelab se ve que se pueden definir valores predeterminados para los parámetros de una función, como se observa en la función “birthdayGreeting” donde el parámetro “name” tiene un valor predeterminado de 'Rover'. Esto permite omitir el argumento correspondiente al llamar a la función, como en la primera llamada dentro de la función main donde solo se proporciona la edad, y el nombre toma automáticamente su valor por defecto.

En este caso se define una función llamada “birthdayGreeting” como se mencionaba anteriormente que toma un nombre (con un valor predeterminado de "Rover") y una edad como entrada, y devuelve una cadena formateada con un mensaje de cumpleaños personalizado. La función principal (main) llama a birthdayGreeting dos veces: la primera vez proporcionando solo la edad (5), lo que hace que se utilice el nombre por defecto "Rover", y la segunda vez proporcionando tanto el nombre ("Rex") como la edad (2). Las cadenas de cumpleaños resultantes de estas llamadas se imprimen en la consola, mostrando los mensajes "Happy Birthday, ¡Rover! You are now 5 years old!" y "Happy Birthday, Rex! You are now 2 years old!".

The screenshot shows the Kotlin Play IDE interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the navigation bar, there are dropdown menus for '2.1.20' and 'JVM'. A 'Program arguments' input field contains the value 'Rover'. To the right of the input field are 'Copy link', 'Share code', and a purple 'Run' button. The main area displays the following Kotlin code:

```
fun main() {
    println(birthdayGreeting(age = 5))
    println(birthdayGreeting("Rex", 2))
}

fun birthdayGreeting(name: String = "Rover", age: Int): String {
    return "Happy Birthday, $name! You are now $age years old!"
}
```

Below the code, the output window shows the results of the program execution:

```
Happy Birthday, Rover! You are now 5 years old!
Happy Birthday, Rex! You are now 2 years old!
```

Ilustración 6 Argumentos predeterminados

### Problemas prácticos: Conceptos básicos de Kotlin

Se muestra cómo imprimir mensajes en consola utilizando `println()`, así como el uso de plantillas de cadenas, lo cual permite insertar valores de variables dentro de una cadena de texto de manera directa utilizando el símbolo “\$”, lo que hace que el código sea más limpio y legible. Se profundiza también en la concatenación de cadenas, una operación común en la programación, y cómo el uso de operadores como “+” que puede ayudar a combinar textos y variables.

The screenshot shows the Kotlin Play IDE interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the navigation bar, there are dropdown menus for '2.1.20' and 'JVM'. A 'Program arguments' input field contains the value 'Rover'. To the right of the input field are 'Copy link', 'Share code', and a purple 'Run' button. The main area displays the following Kotlin code:

```
fun main() {
    println("Use the val keyword when the value doesn't change.")
    println("Use the var keyword when the value can change.")
    println("When you define a function, you define the parameters that can be passed to it.")
    println("When you call a function, you pass arguments for the parameters.")
}
```

Below the code, the output window shows the results of the program execution:

```
Use the val keyword when the value doesn't change.
Use the var keyword when the value can change.
When you define a function, you define the parameters that can be passed to it.
When you call a function, you pass arguments for the parameters.
```

Ilustración 7 Impresión de Mensajes

The screenshot shows the Kotlin Play IDE interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the navigation bar, there are dropdown menus for version (2.1.20) and JVM, and a "Program arguments" input field containing the text "New chat message from a friend". To the right of these are buttons for Copy link, Share code, and Run. The main workspace displays the following Kotlin code:

```
fun main() {
    println("New chat message from a friend")
}
```

Below the code, the output window shows the result of the execution:

```
New chat message from a friend
```

Ilustración 8 Corrección de Error de Compilación

The screenshot shows the Kotlin Play IDE interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the navigation bar, there are dropdown menus for version (2.1.20) and JVM, and a "Program arguments" input field containing the text "Sale - Up to \$discountPercentage% discount off \$item! Hurry Up!". To the right of these are buttons for Copy link, Share code, and Run. The main workspace displays the following Kotlin code:

```
fun main() {
    val discountPercentage = 20
    val item = "Google Chromecast"
    val offer = "Sale - Up to $discountPercentage% discount off $item! Hurry Up!"

    println(offer)
}
```

Below the code, the output window shows the result of the execution:

```
Sale - Up to 20% discount off Google Chromecast! Hurry Up!
```

Ilustración 9 Plantillas de Cadenas

The screenshot shows the Kotlin Play IDE interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the navigation bar, there are dropdown menus for version (2.1.20) and JVM, and a "Program arguments" input field. On the right side of the toolbar are buttons for "Copy link", "Share code", and a purple "Run" button. The main code editor area contains the following Kotlin code:

```
fun main() {
    val numberOfAdults = 20
    val numberOfKids = 30
    val total = numberOfAdults + numberOfKids
    println("The total party size is: $total")
}
```

When the code is run, the output window displays the result: "The total party size is: 50".

Ilustración 10 Concatenación de Cadenas

The screenshot shows the Kotlin Play IDE interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the navigation bar, there are dropdown menus for version (2.1.20) and JVM, and a "Program arguments" input field. On the right side of the toolbar are buttons for "Copy link", "Share code", and a purple "Run" button. The main code editor area contains the following Kotlin code:

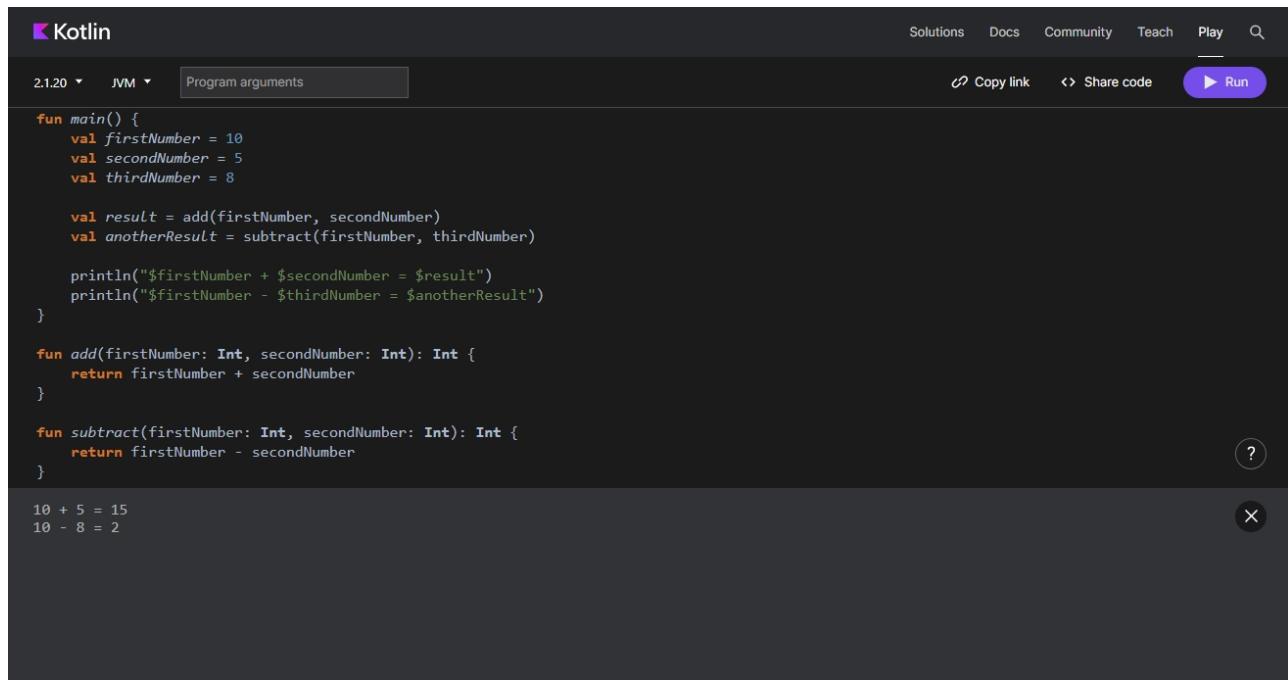
```
fun main() {
    val baseSalary = 5000
    val bonusAmount = 1000
    val totalSalary = "$baseSalary + $bonusAmount"
    println("Congratulations for your bonus! You will receive a total of $totalSalary (additional bonus).")
}
```

When the code is run, the output window displays the result: "Congratulations for your bonus! You will receive a total of 5000 + 1000 (additional bonus.)."

Ilustración 11 Formato de Mensajes

También se explora cómo realizar operaciones matemáticas básicas, como suma y resta, y cómo encapsular estas operaciones en funciones para reutilizarlas. Además, se presenta la creación de funciones con parámetros predeterminados, lo que permite que ciertos argumentos tengan valores por defecto si no se especifican al llamar a la función.

Se destaca también la importancia de seguir las convenciones de nomenclatura de Kotlin, como usar notación camelCase para las variables y funciones, lo cual mejora la claridad y mantenibilidad del código. Se toca el tema de la comparación de valores, utilizando operadores como “>”, y cómo Kotlin evalúa las comparaciones como valores booleanos.



The screenshot shows the Kotlin playground interface. At the top, there are dropdown menus for 'Solutions', 'Docs', 'Community', 'Teach', 'Play', and a search bar. Below the menu is a toolbar with 'Copy link', 'Share code', and a 'Run' button. The main area contains the following Kotlin code:

```
fun main() {
    val firstNumber = 10
    val secondNumber = 5
    val thirdNumber = 8

    val result = add(firstNumber, secondNumber)
    val anotherResult = subtract(firstNumber, thirdNumber)

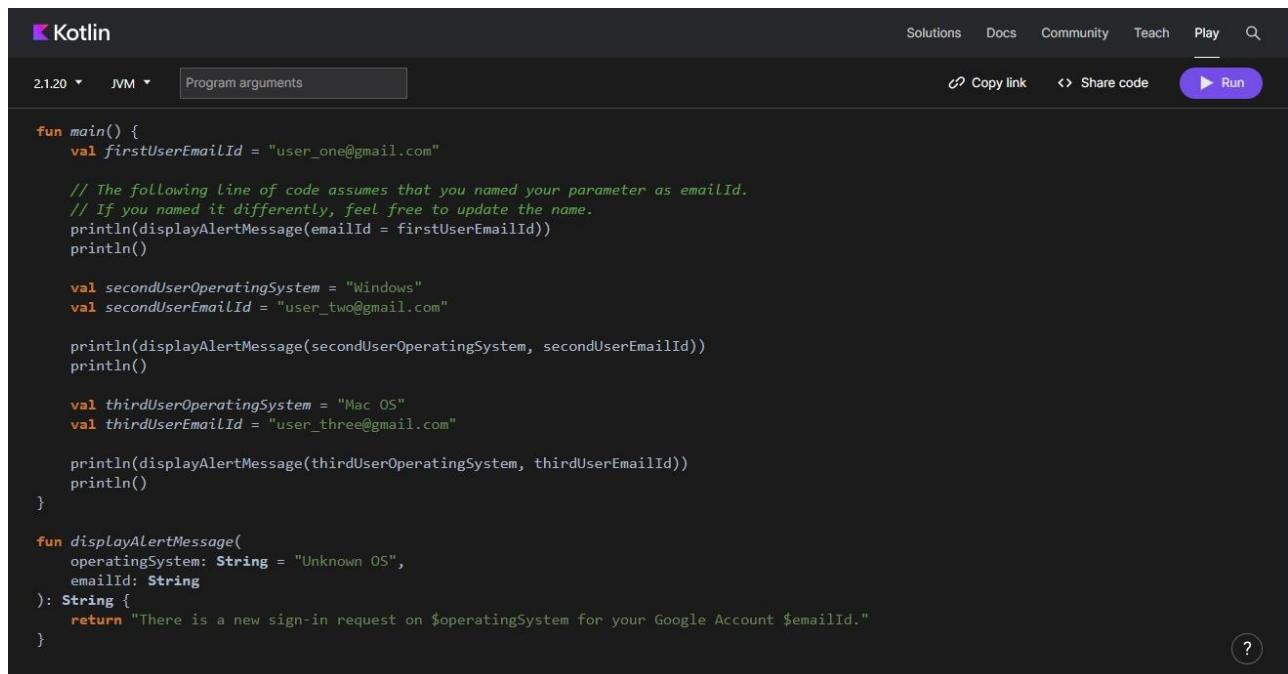
    println("$firstNumber + $secondNumber = $result")
    println("$firstNumber - $thirdNumber = $anotherResult")
}

fun add(firstNumber: Int, secondNumber: Int): Int {
    return firstNumber + secondNumber
}

fun subtract(firstNumber: Int, secondNumber: Int): Int {
    return firstNumber - secondNumber
}
```

At the bottom of the code editor, the output of the program is displayed: "10 + 5 = 15" and "10 - 8 = 2".

Ilustración 12 Implementación de Operaciones Matemáticas



The screenshot shows the Kotlin playground interface. At the top, there are dropdown menus for 'Solutions', 'Docs', 'Community', 'Teach', 'Play', and a search bar. Below the menu is a toolbar with 'Copy link', 'Share code', and a 'Run' button. The main area contains the following Kotlin code:

```
fun main() {
    val firstUserEmailId = "user_one@gmail.com"

    // The following line of code assumes that you named your parameter as emailId.
    // If you named it differently, feel free to update the name.
    println(displayAlertMessage(emailId = firstUserEmailId))
    println()

    val secondUserOperatingSystem = "Windows"
    val secondUserEmailId = "user_two@gmail.com"

    println(displayAlertMessage(secondUserOperatingSystem, secondUserEmailId))
    println()

    val thirdUserOperatingSystem = "Mac OS"
    val thirdUserEmailId = "user_three@gmail.com"

    println(displayAlertMessage(thirdUserOperatingSystem, thirdUserEmailId))
    println()
}

fun displayAlertMessage(
    operatingSystem: String = "Unknown OS",
    emailId: String
): String {
    return "There is a new sign-in request on $operatingSystem for your Google Account $emailId."
}
```

Ilustración 13 Parámetros Predeterminados

The screenshot shows the Kotlin playground interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the navigation bar, the version is set to 2.1.20 and the JVM target is selected. A "Program arguments" input field contains no text. To the right of the input field are buttons for "Copy link", "Share code", and a purple "Run" button. The main area displays the following Kotlin code:

```
fun main() {
    val firstUserEmailId = "user_one@gmail.com"

    // The following line of code assumes that you named your parameter as emailId.
    // If you named it differently, feel free to update the name.
    println(displayAlertMessage(emailId = firstUserEmailId))
    println()

    val secondUserOperatingSystem = "Windows"
    val secondUserEmailId = "user_two@gmail.com"

    println(displayAlertMessage(secondUserOperatingSystem, secondUserEmailId))
}
```

Below the code, three messages are displayed in a log-like format:

- There is a new sign-in request on Unknown OS for your Google Account user\_one@gmail.com.
- There is a new sign-in request on Windows for your Google Account user\_two@gmail.com.
- There is a new sign-in request on Mac OS for your Google Account user\_three@gmail.com.

Ilustración 7. 1 Parámetros Predeterminados (Ejecución)

En este codelab también se representó un podómetro, que es un dispositivo usado para contar los pasos que una persona da al caminar, comúnmente integrado en teléfonos y relojes inteligentes. El código calcula la cantidad de calorías quemadas a partir del número de pasos dados (en este caso, 4,000), multiplicando ese valor por 0.04 (una estimación de calorías quemadas por paso). Luego, se imprime el resultado.

The screenshot shows the Kotlin playground interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the navigation bar, the version is set to 2.1.20 and the JVM target is selected. A "Program arguments" input field contains the value "4000". To the right of the input field are buttons for "Copy link", "Share code", and a purple "Run" button. The main area displays the following Kotlin code:

```
fun main() {
    val steps = 4000
    val caloriesBurned = pedometerStepsToCalories(steps)
    println("Walking $steps steps burns $caloriesBurned calories")
}

fun pedometerStepsToCalories(numberOfSteps: Int): Double {
    val caloriesBurnedForEachStep = 0.04
    val totalCaloriesBurned = numberOfSteps * caloriesBurnedForEachStep
    return totalCaloriesBurned
}
```

Below the code, the output of the program is shown: "Walking 4000 steps burns 160.0 calories".

Ilustración 14 Podómetro

The screenshot shows the Kotlin playground interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the navigation bar, the version is set to 2.1.20 and the JVM target is selected. A "Program arguments" input field is empty. To the right of the input field are buttons for "Copy link", "Share code", and a purple "Run" button. The main area displays the following Kotlin code:

```
fun main() {
    println("Have I spent more time using my phone today: ${compareTime(300, 250)}")
    println("Have I spent more time using my phone today: ${compareTime(300, 300)}")
    println("Have I spent more time using my phone today: ${compareTime(200, 220)}")
}

fun compareTime(timeSpentToday: Int, timeSpentYesterday: Int): Boolean {
    return timeSpentToday > timeSpentYesterday
}
```

Below the code, the output of the program is shown: "Have I spent more time using my phone today: true", "Have I spent more time using my phone today: false", and "Have I spent more time using my phone today: false".

Ilustración 15 Comparación de dos números

Por último, se enseña a mover código duplicado a funciones, lo que mejora la reutilización y organización del código. La centralización de tareas repetitivas en una sola función permite que, si se necesita modificar la lógica, solo sea necesario hacerlo en un único lugar, evitando la duplicación de código y mejorando la eficiencia del mantenimiento del proyecto.

The screenshot shows a Kotlin code editor interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the bar, there are dropdown menus for '2.1.20' and 'JVM', and a 'Program arguments' input field. To the right are 'Copy link', 'Share code', and a 'Run' button. The main area contains two functions: 'main()' and 'printWeatherForCity()'. The 'main()' function calls 'printWeatherForCity()' for four cities. The 'printWeatherForCity()' function prints the city name, low temperature, high temperature, and chance of rain. The output window below shows the results for Ankara, Tokyo, Cape Town, and Guatemala City.

```
fun main() {
    printWeatherForCity("Ankara", 27, 31, 82)
    printWeatherForCity("Tokyo", 32, 36, 10)
    printWeatherForCity("Cape Town", 59, 64, 2)
    printWeatherForCity("Guatemala City", 50, 55, 7)
}

fun printWeatherForCity(cityName: String, lowTemp: Int, highTemp: Int, chanceOfRain: Int) {
    println("City: $cityName")
    println("Low temperature: $lowTemp, High temperature: $highTemp")
    println("Chance of rain: $chanceOfRain%")
    println()
}
```

```
City: Ankara
Low temperature: 27, High temperature: 31
Chance of rain: 82%

City: Tokyo
Low temperature: 32, High temperature: 36
Chance of rain: 10%

City: Cape Town
Low temperature: 59, High temperature: 64
Chance of rain: 2%
```

Ilustración 16 Reducción de Código Duplicado a Función

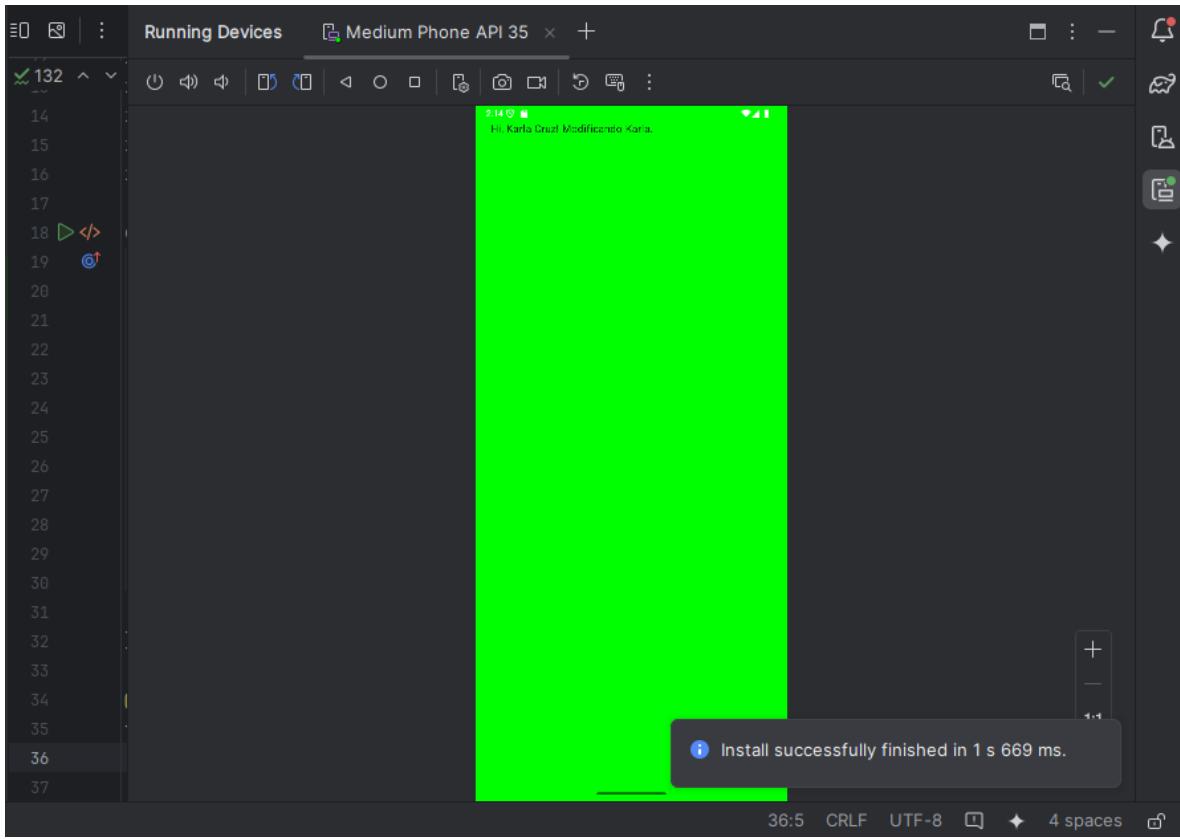
## Ruta de Aprendizaje 2 (Configuración de Android Studio)

### Cómo crear tu primera app para Android

Esta aplicación en Android con Jetpack Compose muestra una tarjeta de saludo simple con un mensaje personalizado. La clase principal “MainActivity” hereda de “ComponentActivity” y sobrescribe el método “onCreate()”. Dentro de este método se utiliza “setContent {}” para definir la interfaz gráfica declarativamente con Compose. La función “GreetingCardTheme” aplica el tema de la app, y dentro de ella, “Surface” actúa como un contenedor de fondo usando el color del tema actual.

La función “Greeting()” es un componente composable, lo que significa que puede usarse para construir la UI de forma modular. Recibe un nombre como parámetro (name: String) y muestra un texto con ese nombre insertado en la cadena. Además, “Surface” dentro de “Greeting()” define un fondo de color cian, y “padding(24.dp)” agrega espacio alrededor del texto.

La función “GreetingPreview()” está anotada con “@Preview”, lo que permite visualizar cómo se verá la UI directamente en el editor de Android Studio, sin necesidad de ejecutar la aplicación en un emulador o dispositivo.



### **Ruta de Aprendizaje 3 (Crear un Diseño Básico)**

#### **Compila una app simple con elementos de texto que admiten composición**

Aquí se define una aplicación Android construida con Jetpack Compose, diseñada para mostrar una tarjeta de cumpleaños con un mensaje personalizado. La clase MainActivity utiliza setContent para declarar toda la interfaz de usuario. Dentro del contenido, se aplica el tema personalizado HappyBirthdayTheme y se usa el componente "Surface" como contenedor de fondo.

La función "GreetingText" es un composable que organiza los textos en una columna (Column), centrando su contenido verticalmente. Muestra dos textos: uno con el mensaje de felicitación y otro con el nombre de quien lo envía. El mensaje tiene un tamaño de fuente grande (100.sp) y está centrado horizontalmente (textAlign = TextAlign.Center), mientras que el nombre del remitente se alinea a la derecha (align(alignment = Alignment.End)) y tiene un tamaño de fuente más pequeño (36.sp).

La función "BirthdayCardPreview" permite previsualizar la tarjeta en el editor de Android Studio usando la anotación "@Preview". Sin embargo a esta le realizamos algunas modificaciones como el texto y el cambiar el fondo de la imagen como se muestra a continuación en el siguiente apartado.

## Agrega imágenes a tu app para Android

Este codelab nos enseña cómo agregar imágenes a una aplicación Android utilizando Jetpack Compose. Se aprende a integrar una imagen en la app utilizando la función “Image”, y cómo ajustar su tamaño con el parámetro “ContentScale.Crop” para que ocupe toda la pantalla mientras mantiene la relación de aspecto. Además, nos muestra cómo cambiar la opacidad de la imagen usando el parámetro “Alpha”, configurándolo en 0.5F para mejorar el contraste.

Por otra parte, se aborda cómo usar modificadores en Jetpack Compose para personalizar los elementos de la interfaz, como el fondo de un “Text” y la disposición de los elementos dentro de un “Row” o “Column”. También se enfoca en cómo usar “verticalArrangement” y “horizontalArrangement” para controlar el posicionamiento de los elementos secundarios, y cómo estos pueden ayudar a organizar el diseño de manera eficiente.

Para este capítulo usamos las ramas de HP\_v1\_diseño y HB\_v1\_comentarios. Las versiones son las siguientes:

Modificación realizada	Versión
Proyecto recién subido	06d4deb
Diseño Karla sin comentarios de Yahir	3ad2e02
Comentarios de Yahir	eedf3b8
Proyecto actual de fusión del diseño Karla con el de Yahir	ca24ebf (main)

El proyecto recién subido (versión 06d4deb) se veía de la siguiente manera:

```
MINGW64:/c/Users/Karla Cruz/AndroidStudioProjects/ProyectosCompartidos/HappyBirthday (main)
Karla_Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos/HappyBirthday
$ git checkout 06d4deb
Updating files: 100% (7388/7388), done.
Note: switching to '06d4deb'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

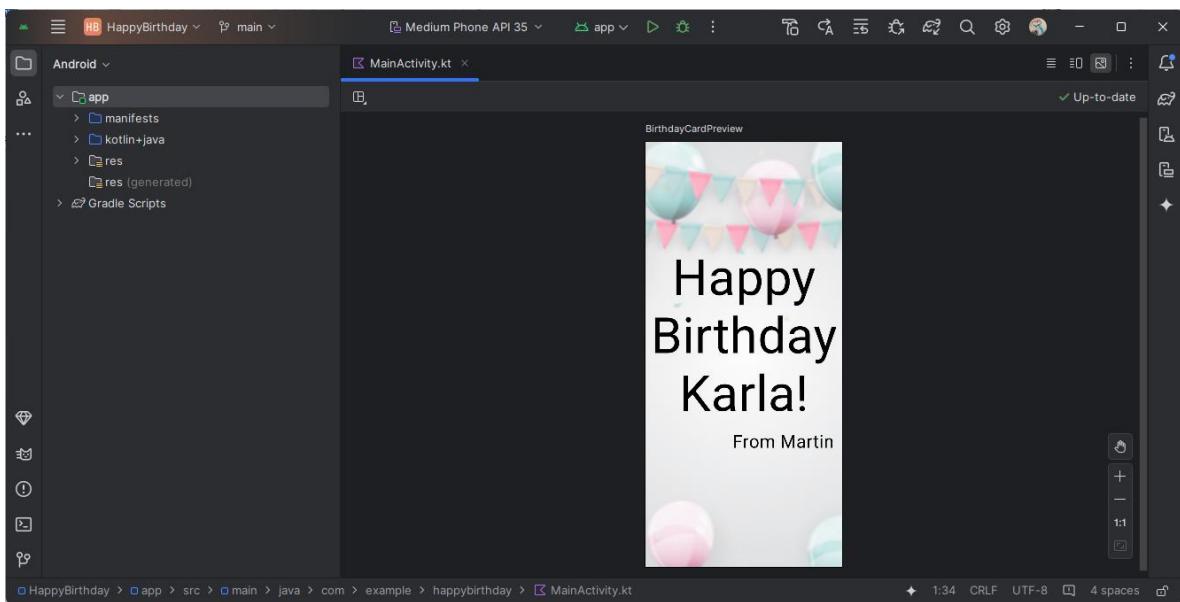
Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false
HEAD is now at 06d4deb Agregado nuevo proyecto HappyBirthday
Karla_Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos/HappyBirthday
```

The screenshot shows the Android Studio interface with the code editor open to `MainActivity.kt`. The code imports various AndroidX components and defines a `MainActivity` class that overrides `onCreate` to set the content to a `HappyBirthdayTheme`.

```
1 package com.example.happybirthday
2
3 import android.os.Bundle
4 import androidx.activity.ComponentActivity
5 import androidx.activity.compose.setContent
6 import com.example.happybirthday.ui.theme.HappyBirthdayTheme
7 import androidx.compose.foundation.layout.*
8 import androidx.compose.material3.*
9 import androidx.compose.runtime.Composable
10 import androidx.compose.ui.Alignment
11 import androidx.compose.ui.Modifier
12 import androidx.compose.foundation.Image // Nuevo import
13 import androidx.compose.ui.layout.ContentScale
14 import androidx.compose.ui.res.painterResource // Nuevo import
15 import androidx.compose.ui.text.style.TextAlign
16 import androidx.compose.ui.tooling.preview.Preview
17 import androidx.compose.ui.unit.dp
18 import androidx.compose.ui.unit.sp
19
20
21 class MainActivity : ComponentActivity() {
22     override fun onCreate(savedInstanceState: Bundle?) {
23         super.onCreate(savedInstanceState)
24         setContent {
25             HappyBirthdayTheme {
26                 ...
27             }
28         }
29     }
30 }
```

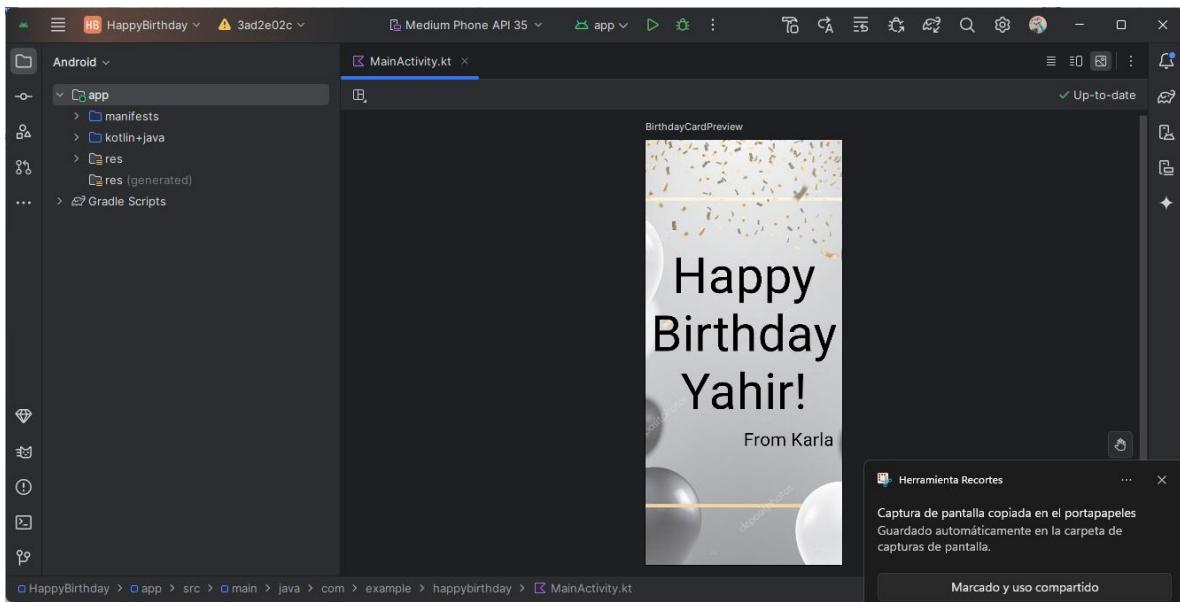


Se le agregó un cambio de diseño a la versión original o recién subida (versión 3ad2e02):

```
Karla.Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos/HappyBirthday ((06d4deb...))
$ git checkout 3ad2e02
Previous HEAD position was 06d4deb Agregado nuevo proyecto HappyBirthday
HEAD is now at 3ad2e02 Modificando texto e imagen en la interfaz de HappyBirthday
```

The screenshot shows the Android Studio interface with the code editor open to `MainActivity.kt`. The code defines a `ComponentActivity` named `MainActivity` that sets its content to a `HappyBirthdayTheme`.

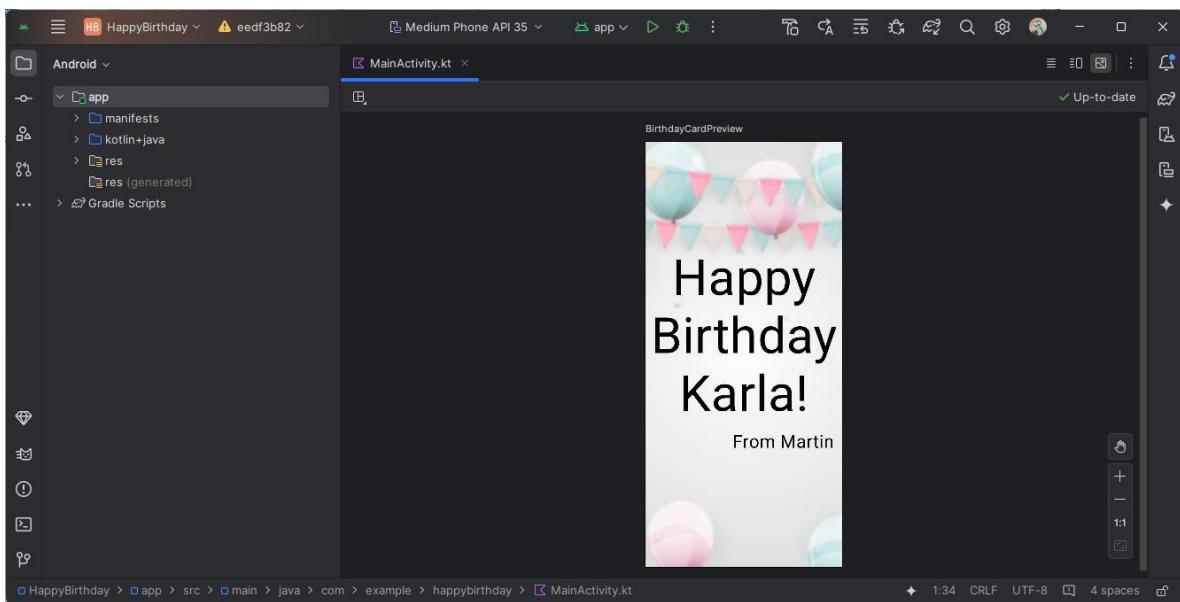
```
1 package com.example.happybirthday
2
3 import android.os.Bundle
4 import androidx.activity.ComponentActivity
5 import androidx.activity.compose.setContent
6 import com.example.happybirthday.ui.theme.HappyBirthdayTheme
7 import androidx.compose.foundation.layout.*
8 import androidx.compose.material3.*
9 import androidx.compose.runtime.Composable
10 import androidx.compose.ui.Alignment
11 import androidx.compose.ui.Modifier
12 import androidx.compose.foundation.Image // Nuevo import
13 import androidx.compose.ui.layout.ContentScale
14 import androidx.compose.ui.res.painterResource // Nuevo import
15 import androidx.compose.ui.text.style.TextAlign
16 import androidx.compose.ui.tooling.preview.Preview
17 import androidx.compose.ui.unit.dp
18 import androidx.compose.ui.unit.sp
19
20
21 class MainActivity : ComponentActivity() {
22     override fun onCreate(savedInstanceState: Bundle?) {
23         super.onCreate(savedInstanceState)
24         setContent {
25             HappyBirthdayTheme {
```



Se agregaron comentarios en la rama de comentarios al proyecto original (versión `eedf3b8`):

```
1 package com.example.happybirthday // Define el paquete de la aplicación
2
3 import android.os.Bundle // Importaciones necesarias para la actividad y el diseño en Jetpack Compose
4 import androidx.activity.ComponentActivity
5 import androidx.activity.compose.setContent
6 import com.example.happybirthday.ui.theme.HappyBirthdayTheme // Importa el tema de la aplicación
7 import androidx.compose.foundation.layout.* // Importaciones para la disposición y el diseño de la interfaz
8 import androidx.compose.material3.* // Importa Material 3 para los componentes visuales
9 import androidx.compose.runtime.Composable
10 import androidx.compose.ui.Alignment
11 import androidx.compose.ui.Modifier
12 import androidx.compose.foundation.Image // Importaciones adicionales para manejar imágenes
13 import androidx.compose.ui.layout.ContentScale
14 import androidx.compose.ui.res.painterResource
15 import androidx.compose.ui.text.style.TextAlign // Importaciones para el formato del texto
16 import androidx.compose.ui.tooling.preview.Preview
17 import androidx.compose.ui.unit.dp // Define medidas en dp (density-independent pixels)
18 import androidx.compose.ui.unit.sp // Define tamaños de fuente en sp (scale-independent pixels)
19
20
21 class MainActivity : ComponentActivity() { // Clase principal de la aplicación
22     override fun onCreate(savedInstanceState: Bundle?) {
23         super.onCreate(savedInstanceState)
24         setContent { // Establece el contenido de la interfaz con Jetpack Compose
25             HappyBirthdayTheme { // Aplica el tema de la aplicación
26                 ...
27             }
28         }
29     }
30 }
31
32 
```

HappyBirthday > app > src > main > java > com > example > happybirthday > MainActivity.kt

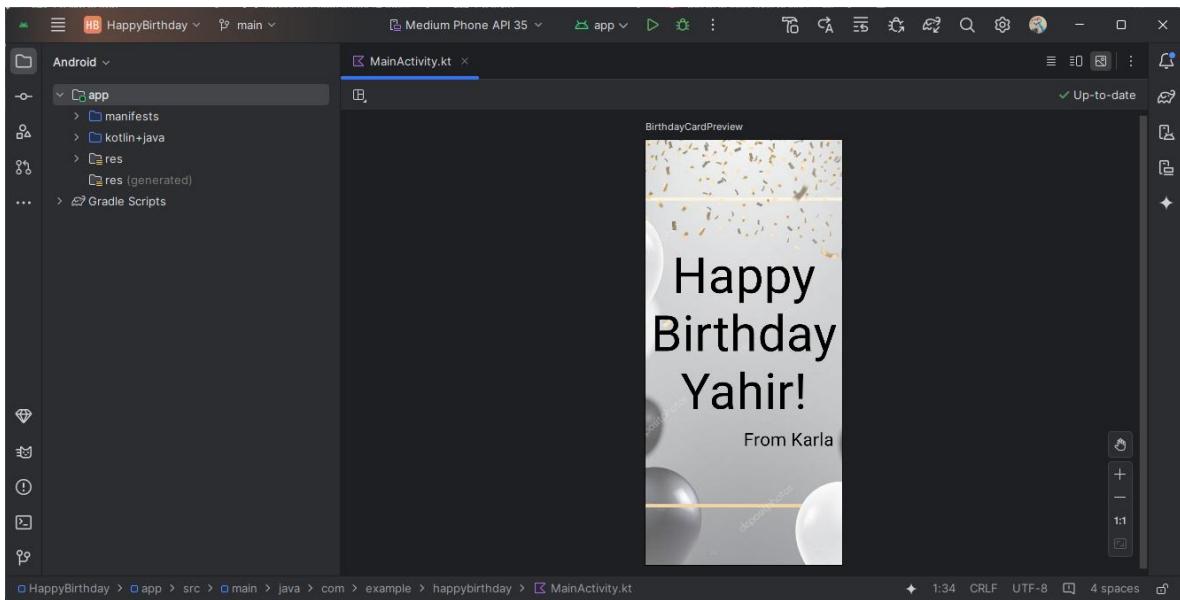


Se fusionaron esos cambios del diseño y los comentarios con la rama principal y ahora el proyecto actual está en la rama main y se ve de la siguiente manera:

The screenshot shows the Android Studio interface with the code editor open to `MainActivity.kt`. The code defines a `ComponentActivity` named `MainActivity` that sets its content to `HappyBirthdayTheme`.

```
package com.example.happybirthday // Define el paquete de la aplicación
import android.os.Bundle // Importaciones necesarias para la actividad y el diseño en Jetpack Compose
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import com.example.happybirthday.ui.theme.HappyBirthdayTheme // Importa el tema de la aplicación
import androidx.compose.foundation.layout.* // Importaciones para la disposición y el diseño de la interfaz
import androidx.compose.material3.* // Importa Material 3 para los componentes visuales
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.foundation.Image // Importaciones adicionales para manejar imágenes
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.style.TextAlign // Importaciones para el formato del texto
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp // Define medidas en dp (density-independent pixels)
import androidx.compose.ui.unit.sp // Define tamaños de fuente en sp (scale-independent pixels)

class MainActivity : ComponentActivity() { // Clase principal de la aplicación
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent { // Establece el contenido de la interfaz con Jetpack Compose
            HappyBirthdayTheme { // Aplica el tema de la aplicación
                ...
            }
        }
    }
}
```

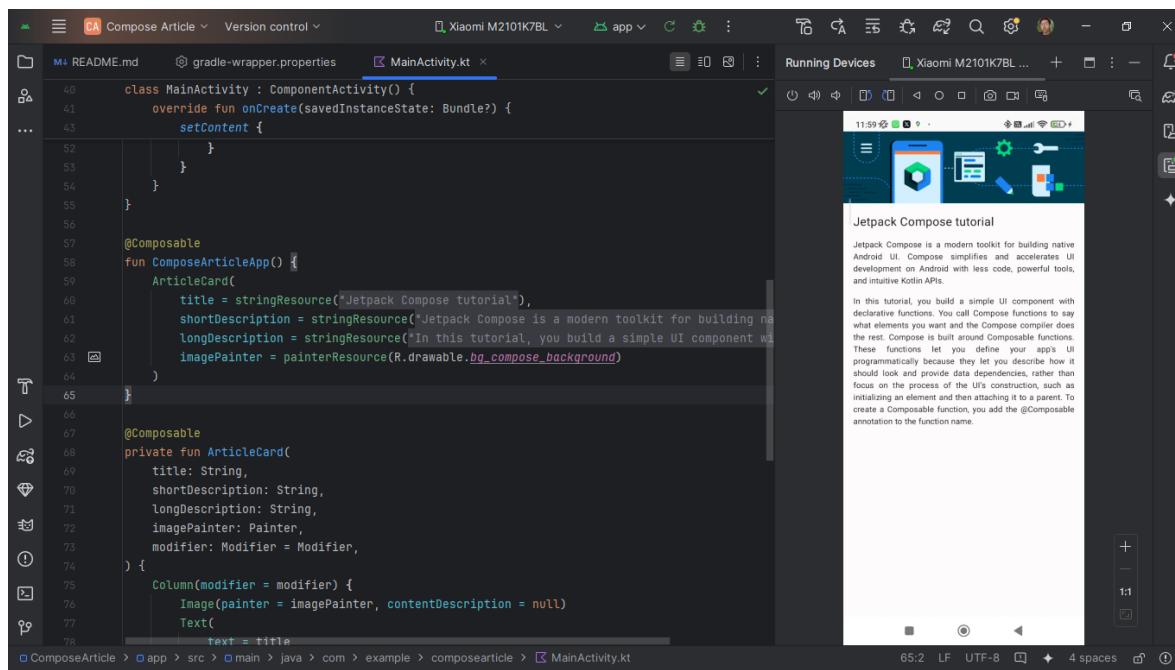


## Problemas prácticos: Conceptos básicos de Compose

Para el artículo de Compose nos centramos en la creación de una pantalla en la aplicación “Learn Together”, que presenta un artículo sobre Jetpack Compose. Se proporciona una estructura para crear una interfaz de usuario utilizando componentes composable de Jetpack Compose, donde en la clase “MainActivity”, se configura el contenido de la actividad utilizando “setContent”, donde se aplica el tema “ComposeArticleTheme” y se llama a la función “ComposeArticleApp()”, que a su vez muestra un artículo con el título, descripción corta y larga, y una imagen de fondo.

La función “ComposeArticleApp” es responsable de pasar los recursos de texto e imagen a la función “ArticleCard”, que organiza estos elementos en una columna vertical (Column). Dentro de esta columna, se coloca una imagen que ocupa todo el ancho de la pantalla, seguida de tres elementos de texto: el título con un tamaño de fuente de 24sp y padding de 16dp, la descripción corta y larga con un texto justificado y padding de 16dp en los lados y en la parte superior e inferior.

El uso de “@Composable” permite que las funciones como “ComposeArticleApp” y “ArticleCard” definan la UI de manera declarativa. Cada componente composable gestiona su propio diseño y disposición. Además, la función “@Preview” permite ver la interfaz en Android Studio sin necesidad de ejecutar la aplicación completa.



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

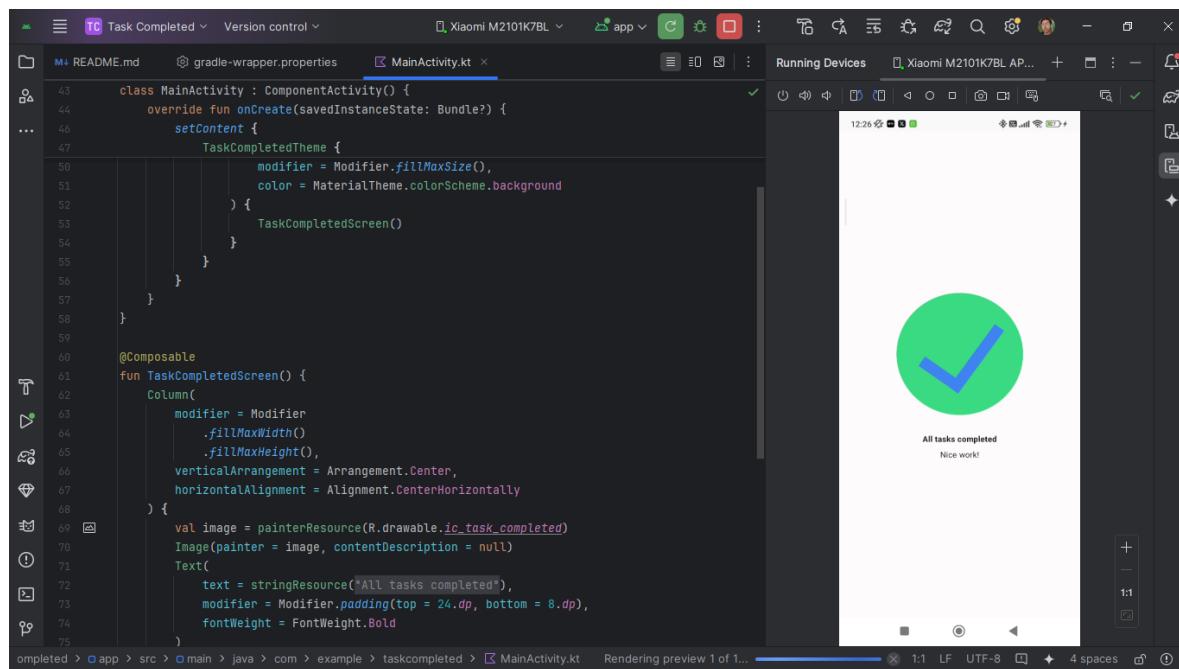
- Project Structure:** Shows files like README.md, gradle-wrapper.properties, and MainActivity.kt.
- MainActivity.kt Code:**

```
40 class MainActivity : ComponentActivity() {
41     override fun onCreate(savedInstanceState: Bundle?) {
42         setContent {
43             ...
44         }
45     }
46
47     @Composable
48     fun ComposeArticleApp() {
49         ArticleCard(
50             title = stringResource(R.string.compose_tutorial),
51             shortDescription = stringResource(R.string.compose_tutorial_short_desc),
52             longDescription = stringResource(R.string.compose_tutorial_long_desc),
53             imagePainter = painterResource(R.drawable.bg_compose_background)
54         )
55     }
56
57     @Composable
58     private fun ArticleCard(
59         title: String,
60         shortDescription: String,
61         longDescription: String,
62         imagePainter: Painter,
63         modifier: Modifier = Modifier,
64     ) {
65         Column(modifier = modifier) {
66             Image(painter = imagePainter, contentDescription = null)
67             Text(
68                 text = title
69             )
70             Text(
71                 text = shortDescription
72             )
73             Text(
74                 text = longDescription
75             )
76         }
77     }
78 }
```
- Preview Window:** Displays the “Jetpack Compose tutorial” screen. It includes a title bar with icons, a main content area with a background image, and three text blocks (title, short description, long description) arranged vertically.
- Bottom Status Bar:** Shows device information (Xiaomi M2101K7BL), battery level (11:59), signal strength, and connectivity.

Para el administrador de tareas se construye una pantalla en donde se muestra cuando el usuario ha completado todas sus tareas del día. La interfaz muestra de manera clara y visual que no quedan tareas pendientes. Para ello, se proporcionan recursos como una imagen ilustrativa y dos cadenas de texto: "All tasks completed" y "Nice work!".

En el código se implementa la pantalla usando Jetpack Compose. Dentro de "MainActivity", se configura la UI con "setContent" y se llama a "TaskCompletedApp()", que contiene el contenido principal de la pantalla. El diseño se basa en un Column, centrado tanto vertical como horizontalmente usando "verticalArrangement = Arrangement.Center" y "horizontalAlignment = Alignment.CenterHorizontally", para cumplir con las especificaciones de alineación.

Dentro de esta columna se inserta primero una imagen (Image) usando "painterResource", seguida de dos elementos de texto (Text). El primero, "All tasks completed", se muestra con estilo de fuente en negrita y paddings superior e inferior de 24dp y 8dp, respectivamente. El segundo texto de "Nice work!", se muestra con un tamaño de fuente de 16sp.



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

- Left Panel (Project Tree):** Shows files like README.md, gradle-wrapper.properties, and MainActivity.kt.
- Top Bar:** Displays "Task Completed", "Version control", "Xiaomi M2101K7BL", "app", and other icons.
- Main Area (Code Editor):** Contains the MainActivity.kt code. The code defines a class MainActivity that overrides onCreate and sets the content to a TaskCompletedTheme. This theme uses a Column with vertical arrangement center and horizontal alignment center horizontally. It contains an Image of a checkmark and two Text elements: "All tasks completed" in bold with top padding of 24dp and bottom padding of 8dp, and "Nice work!" in 16sp font.
- Right Panel (Preview):** Shows a rendering of the app on a Xiaomi M2101K7BL device. The screen features a large green circular icon with a blue checkmark, followed by the text "All tasks completed" and "Nice work!".
- Bottom Bar:** Shows the file path "completed > app > src > main > java > com > example > taskcompleted > MainActivity.kt", the rendering preview status "Rendering preview 1 of 1...", and various tool icons.

En cuanto al cuadrante de Compose se construye una pantalla educativa para una app que muestra información sobre funciones “Composable” en Jetpack Compose, dividiendo la interfaz en cuatro cuadrantes iguales. Cada uno representa una función específica: “Text”, “Image”, “Row” y “Column”, e incluye su nombre en texto en negrita seguido de una breve descripción.

El diseño se logra usando un Column principal que contiene dos Row, cada una con dos elementos hijos que representan los cuadrantes. Para dividir la pantalla de manera equitativa, se emplea el modificador “Modifier.weight(1f)” tanto en filas como en columnas, lo que permite que cada sección ocupe el mismo espacio vertical u horizontal según corresponda. Dentro de cada cuadrante, los elementos están alineados al centro en ambas direcciones (“Alignment.CenterHorizontally” y “Arrangement.Center”) y rodeados por un padding de 16dp.

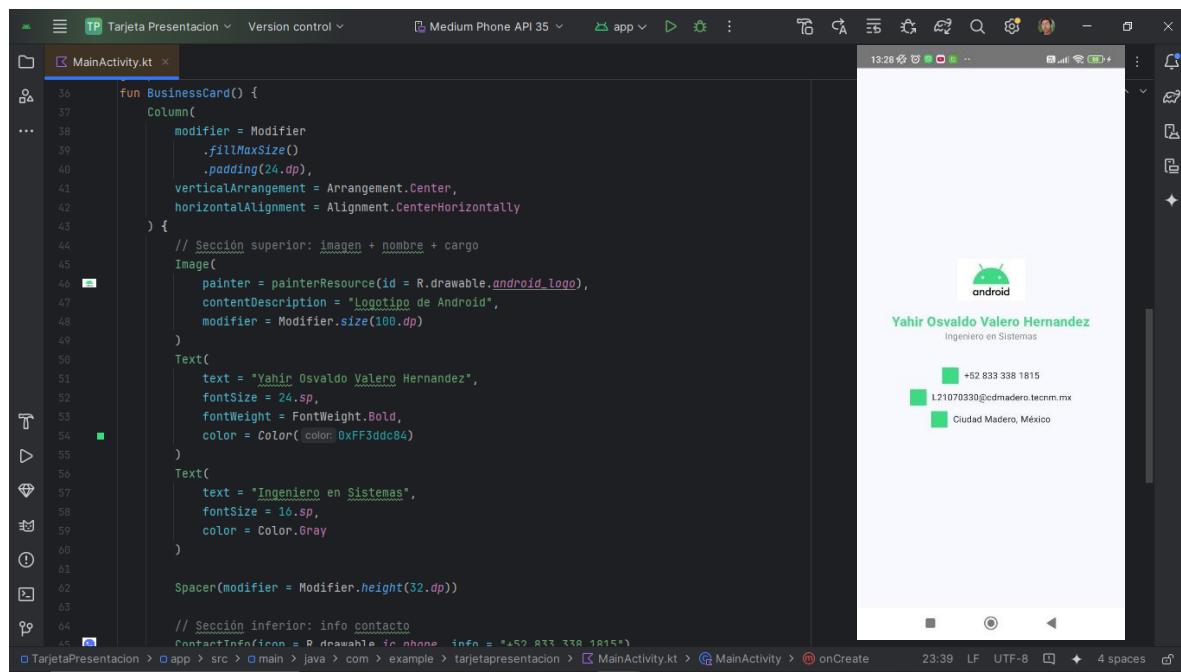
Cada tarjeta (cuadrante) está coloreada con uno de los cuatro colores pastel proporcionados, y contiene dos “Text”: el primero, que es el título (como “Text composable”), se muestra en negrita con padding inferior de 16dp, y el segundo presenta una descripción más larga con fuente por defecto.

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

- Left Panel:** Shows the project structure with files: README.md, gradle-wrapper.properties, and MainActivity.kt.
- Main Editor:** Displays the `MainActivity.kt` code. The code defines a `MainActivity` class that sets its content to a `ComposeQuadrantTheme`. This theme uses a `Surface` modifier with `fillMaxSize()` and `MaterialTheme.colorScheme.background`. It also contains a `ComposeQuadrantApp` function.
- Right Panel:** Shows a preview of the app running on a "Xiaomi M2101K7BL" device. The screen is divided into four quadrants, each containing a card with text and an image. The top-left quadrant is purple and labeled "Text composable". The top-right is purple and labeled "Image composable". The bottom-left is blue and labeled "Row composable". The bottom-right is blue and labeled "Column composable". Each card has descriptive text below it.
- Bottom Status Bar:** Shows file paths, line numbers (51-65), and other development settings.

## Proyecto: Crea una App de Tarjetas de Presentación

Se construye una app simple que simula una tarjeta personal. La interfaz se divide en dos secciones: una superior con el logotipo de Android (u otro), nuestro nombre y cargo profesional, y otra inferior con la información de contacto acompañada de íconos (como teléfono y correo). Se usan elementos como “Text”, “Image”, “Icon”, “Row” y “Column”, organizados mediante modificadores como padding, weight y alignment.



The screenshot shows the Android Studio interface. On the left, the code editor displays `MainActivity.kt` with Kotlin code for a business card application. The code uses Compose UI components like `Column`, `Image`, and `Text` to layout the card. On the right, the preview window shows the final result: a digital business card for "Yahir Osvaldo Valero Hernandez" with contact information including a phone number and email. The preview is set to a medium phone screen with API 35.

```
36    fun BusinessCard() {
37        Column(
38            modifier = Modifier
39                .fillMaxSize()
40                .padding(24.dp),
41            verticalArrangement = Arrangement.Center,
42            horizontalAlignment = Alignment.CenterHorizontally
43        ) {
44            // Sección superior: imagen + nombre + cargo
45            Image(
46                painter = painterResource(id = R.drawable.android_logo),
47                contentDescription = "Logotipo de Android",
48                modifier = Modifier.size(100.dp)
49            )
50            Text(
51                text = "Yahir Osvaldo Valero Hernandez",
52                fontSize = 24.sp,
53                fontWeight = FontWeight.Bold,
54                color = Color(0xFF3ddc84)
55            )
56            Text(
57                text = "Ingeniero en Sistemas",
58                fontSize = 16.sp,
59                color = Color.Gray
60            )
61            Spacer(modifier = Modifier.height(32.dp))
62            // Sección inferior: info contacto
63            ContactInfoIcon = R.drawable.ic_phone_info = "+52 833 333 1215"
64        }
65    }
```

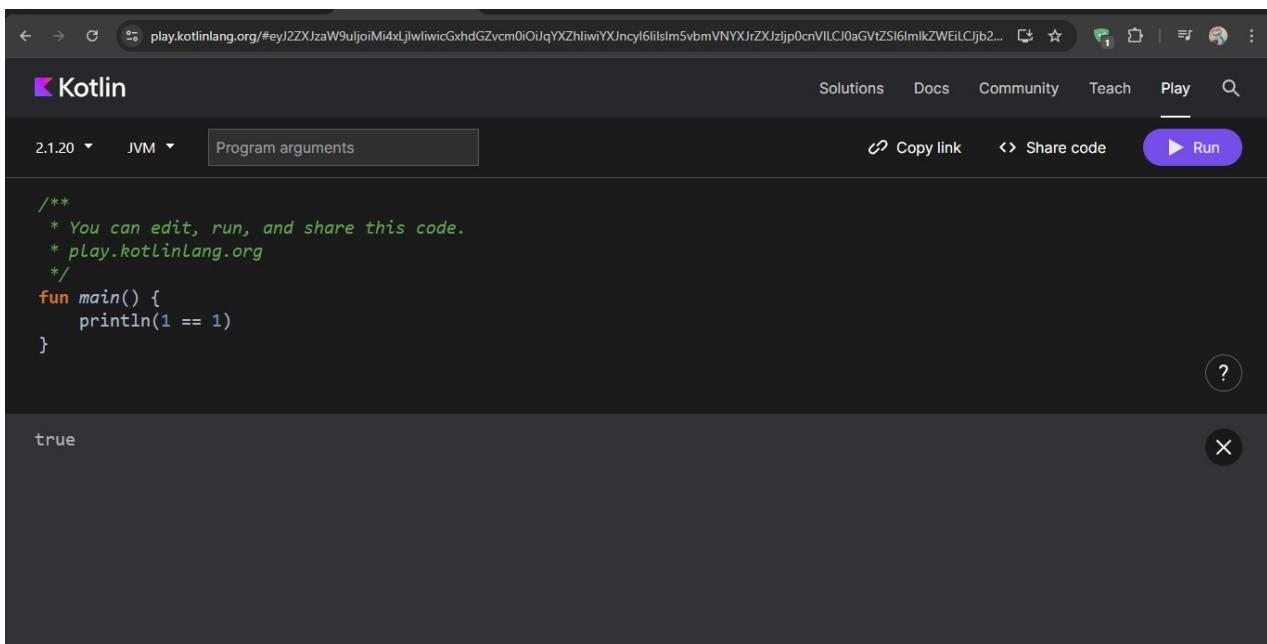
Nota: Los códigos con comentarios se encuentran subidos en github.

## Unidad 2: Creación de una IU de una app

### Ruta de Aprendizaje 1 (Conceptos Básicos de Kotlin)

#### Escribe condicionales en Kotlin

En Kotlin, se pueden usar los condicionales “if/else” o “when” para realizar decisiones basadas en condiciones específicas, donde el condicional if/else funciona evaluando una expresión booleana. El bloque de código dentro de la rama “if” se ejecutará solo cuando la expresión sea verdadera. Si la condición del “if” es falsa, el bloque dentro de la rama “else if” (si existe) se evaluará, y si todas las condiciones anteriores fallan, se ejecutará la rama “else” final.



The screenshot shows the Kotlin Play IDE interface. The code editor contains the following Kotlin code:

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    println(1 == 1)
}
```

The output window below the code editor displays the result of the execution: "true".

Kotlin

Solutions Docs Community Teach Play

2.1.20 JVM Program arguments

Copy link Share code Run

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    println(1 < 1)
}
```

false

Kotlin

Solutions Docs Community Teach Play

2.1.20 JVM Program arguments

Copy link Share code Run

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    val trafficLightColor = "Red"

    if (trafficLightColor == "Red") {
        println("Stop")
    }
}
```

Stop

The screenshot shows the Kotlin Play interface with the following code:

```
2.1.20 ▾ JVM ▾ Program arguments
```

```
fun main() {
    val trafficLightColor = "Red"

    if (trafficLightColor == "Red") {
        println("Stop")
    } else {
        println("Go")
    }
}
```

The output window displays the result:

```
Stop
```

The screenshot shows the Kotlin Play interface with the following code:

```
2.1.20 ▾ JVM ▾ Program arguments
```

```
/*
fun main() {
    val trafficLightColor = "Green"

    if (trafficLightColor == "Red") {
        println("Stop")
    } else {
        println("Go")
    }
}
```

The output window displays the result:

```
Go
```

Kotlin

2.1.20 ▾ JVM ▾ Program arguments

```
/*
 */
fun main() {
    val trafficLightColor = "Yellow"

    if (trafficLightColor == "Red") {
        println("Stop")
    } else if (trafficLightColor == "Yellow") {
        println("Slow")
    } else {
        println("Go")
    }
}

Slow
```

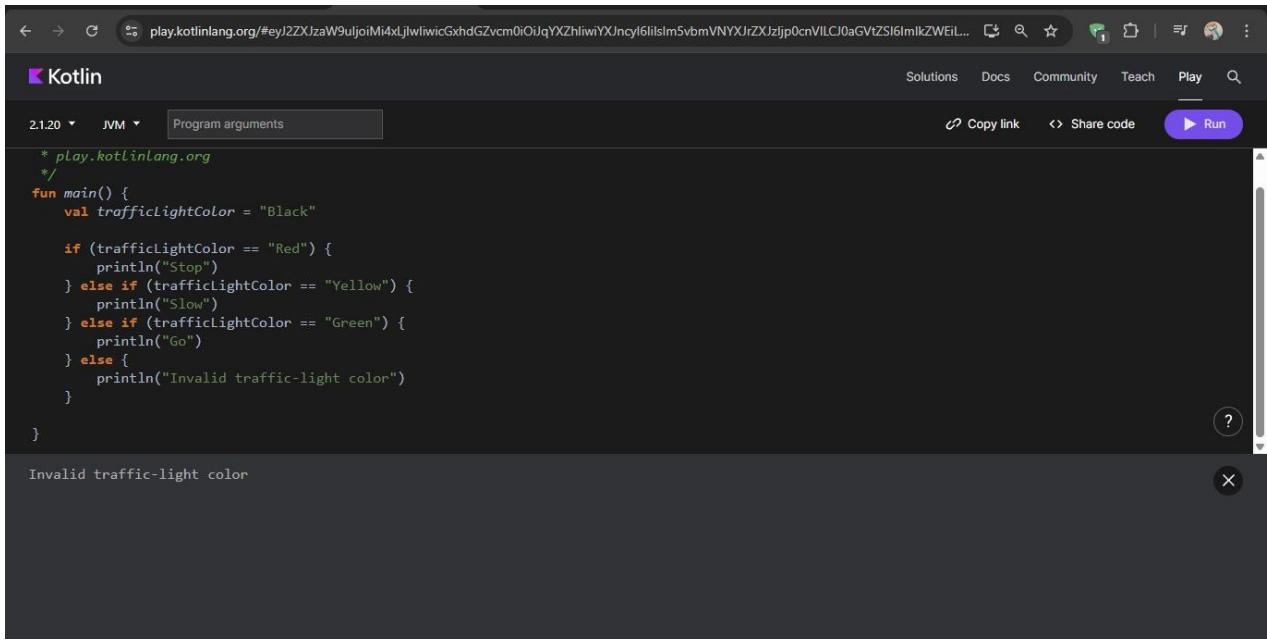
Kotlin

2.1.20 ▾ JVM ▾ Program arguments

```
/*
 */
fun main() {
    val trafficLightColor = "Black"

    if (trafficLightColor == "Red") {
        println("Stop")
    } else if (trafficLightColor == "Yellow") {
        println("Slow")
    } else {
        println("Go")
    }
}

Go
```



The screenshot shows the Kotlin Play IDE interface. The code editor contains the following Kotlin code:

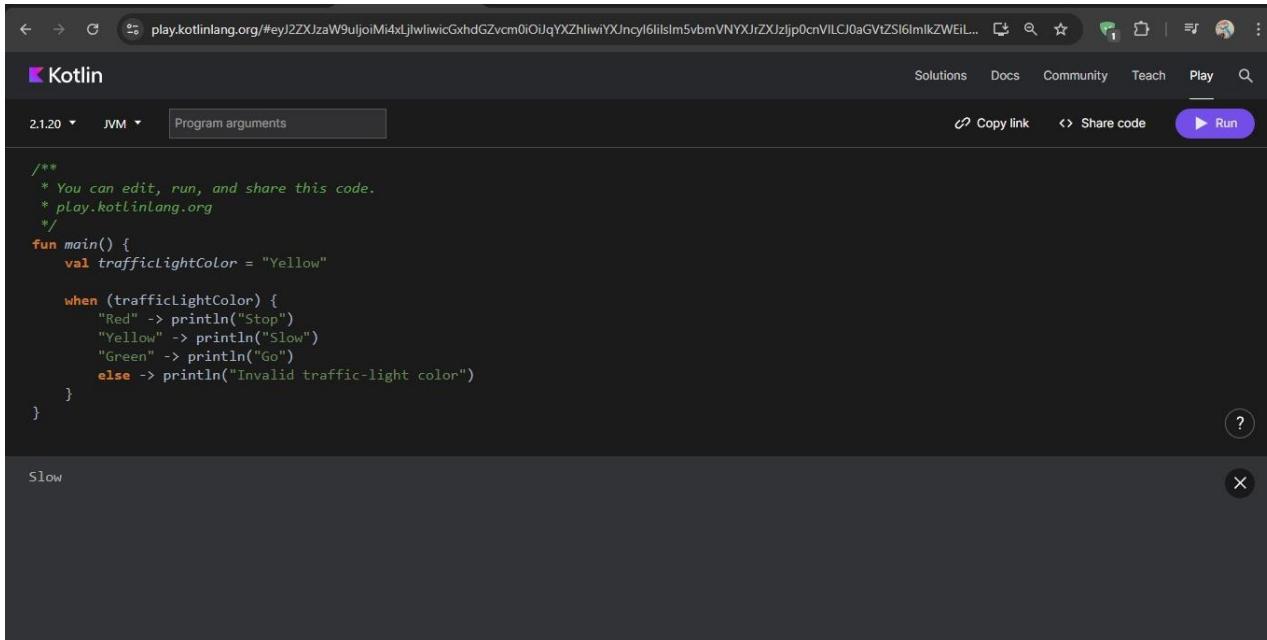
```
* play.kotlinlang.org
*/
fun main() {
    val trafficLightColor = "Black"

    if (trafficLightColor == "Red") {
        println("Stop")
    } else if (trafficLightColor == "Yellow") {
        println("Slow")
    } else if (trafficLightColor == "Green") {
        println("Go")
    } else {
        println("Invalid traffic-light color")
    }
}
```

The output window below the code editor displays the message: "Invalid traffic-light color".

Cuando se tienen más de dos posibles ramas se recomienda utilizar el condicional “when”, ya que facilita la lectura y manejo de múltiples condiciones. “When” puede manejar condiciones más complejas utilizando comas (’,’) para evaluar varias condiciones simultáneamente, rangos con “in”, o la palabra clave “is” para verificar el tipo de una variable.

Además, tanto “if/else” como “when” en Kotlin pueden actuar como sentencias (sin valor de retorno) o como expresiones (que retornan un valor), lo que brinda flexibilidad en su uso.



The screenshot shows the Kotlin Play IDE interface. The code editor contains the following Kotlin code:

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    val trafficLightColor = "Yellow"

    when (trafficLightColor) {
        "Red" -> println("Stop")
        "Yellow" -> println("Slow")
        "Green" -> println("Go")
        else -> println("Invalid traffic-light color")
    }
}
```

The output window below the code editor displays the message: "Slow".

Kotlin

2.1.20 JVM Program arguments

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    val x = 3

    when (x) {
        2 -> println("x is a prime number between 1 and 10.")
        3 -> println("x is a prime number between 1 and 10.")
        5 -> println("x is a prime number between 1 and 10.")
        7 -> println("x is a prime number between 1 and 10.")
        else -> println("x isn't a prime number between 1 and 10.")
    }
}

x is a prime number between 1 and 10.
```

Kotlin

2.1.20 JVM Program arguments

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    val x = 3

    when (x) {
        2, 3, 5, 7 -> println("x is a prime number between 1 and 10.")
        else -> println("x isn't a prime number between 1 and 10.")
    }
}

x is a prime number between 1 and 10.
```

Kotlin

Solutions Docs Community Teach Play

2.1.20 JVM Program arguments

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    val x = 4

    when (x) {
        2, 3, 5, 7 -> println("x is a prime number between 1 and 10.")
        in 1..10 -> println("x is a number between 1 and 10, but not a prime number.")
        else -> println("x isn't a prime number between 1 and 10.")
    }
}
```

x is a number between 1 and 10, but not a prime number.

Kotlin

Solutions Docs Community Teach Play

2.1.20 JVM Program arguments

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    val x: Any = 20

    when (x) {
        2, 3, 5, 7 -> println("x is a prime number between 1 and 10.")
        in 1..10 -> println("x is a number between 1 and 10, but not a prime number.")
        is Int -> println("x is an integer number, but not between 1 and 10.")
        else -> println("x isn't an integer number.")
    }
}
```

x is an integer number, but not between 1 and 10.

Kotlin

Solutions Docs Community Teach Play

2.1.20 JVM Program arguments

Copy link Share code Run

```
/**  
 * You can edit, run, and share this code.  
 * play.kotlinlang.org  
 */  
fun main() {  
    val trafficLightColor = "Amber"  
  
    when (trafficLightColor) {  
        "Red" -> println("Stop")  
        "Yellow", "Amber" -> println("Slow")  
        "Green" -> println("Go")  
        else -> println("Invalid traffic-light color")  
    }  
}  
  
Slow
```

Kotlin

Solutions Docs Community Teach Play

2.1.20 JVM Program arguments

Copy link Share code Run

```
/**  
 * You can edit, run, and share this code.  
 * play.kotlinlang.org  
 */  
fun main() {  
    val trafficLightColor = "Black"  
  
    val message =  
        if (trafficLightColor == "Red") "Stop"  
        else if (trafficLightColor == "Yellow") "Slow"  
        else if (trafficLightColor == "Green") "Go"  
        else "Invalid traffic-light color"  
  
    println(message)  
}  
  
Invalid traffic-light color
```

The screenshot shows a web-based Kotlin playground. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the bar, the title 'Kotlin' is displayed, followed by version '2.1.20' and 'JVM'. A 'Program arguments' input field is present. On the right, there are buttons for 'Copy link', 'Share code', and 'Run'.

The main area contains the following Kotlin code:

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    val trafficLightColor = "Amber"

    val message = when(trafficLightColor) {
        "Red" -> "Stop"
        "Yellow", "Amber" -> "Proceed with caution."
        "Green" -> "Go"
        else -> "Invalid traffic-light color"
    }
    println(message)
}
```

The output of the code is displayed in the terminal below:

```
Proceed with caution.
```

## Usa la nulabilidad en Kotlin

En Kotlin, las variables pueden ser anulables, lo que significa que pueden contener “null”. Las variables no anulables no pueden tener “null”.

The screenshot shows a web-based Kotlin playground. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the bar, the title 'Kotlin' is displayed, followed by version '2.1.20' and 'JVM'. A 'Program arguments' input field is present. On the right, there are buttons for 'Copy link', 'Share code', and 'Run'.

The main area contains the following Kotlin code:

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    val favoriteActor = null
    println(favoriteActor)
}
```

The output of the code is displayed in the terminal below:

```
null
```

The screenshot shows the Kotlin Play IDE interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the navigation bar is a toolbar with dropdowns for '2.1.20' and 'JVM', a 'Program arguments' input field, and buttons for 'Copy link', 'Share code', and 'Run'. The main area contains the following Kotlin code:

```
/**  
 * You can edit, run, and share this code.  
 * play.kotlinlang.org  
 */  
fun main() {  
    var favoriteActor: String? = "Sandra Oh"  
    println(favoriteActor)  
  
    favoriteActor = null  
    println(favoriteActor)  
}
```

Below the code, the output window displays the results of the program execution:

```
Sandra Oh  
null
```

The screenshot shows the Kotlin Play IDE interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the navigation bar is a toolbar with dropdowns for '2.1.20' and 'JVM', a 'Program arguments' input field, and buttons for 'Copy link', 'Share code', and 'Run'. The main area contains the following Kotlin code:

```
/**  
 * You can edit, run, and share this code.  
 * play.kotlinlang.org  
 */  
fun main() {  
    var number: Int? = 10  
    println(number)  
  
    number = null  
    println(number)  
}
```

Below the code, the output window displays the results of the program execution:

```
10  
null
```

The screenshot shows the Kotlin Play IDE interface. The code editor contains the following Kotlin code:

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    var favoriteActor: String = "Sandra Oh"
    println(favoriteActor.length)
}
```

The output window below the code editor shows the result of running the program: "9".

Para acceder a propiedades de variables anulables sin errores, se usan operadores como “?.” (llamada segura) o “!!” (aserción no nula).

The screenshot shows the Kotlin Play IDE interface. The code editor contains the following Kotlin code, with a nullable variable type (String?) instead of a non-nullable one (String):

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    var favoriteActor: String? = null
    println(favoriteActor?.length)
}
```

The output window below the code editor shows the result of running the program: "null".

The screenshot shows the Kotlin Play IDE interface. The code editor contains the following Kotlin code:

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    var favoriteActor: String? = "Sandra Oh"
    println(favoriteActor!!.length)
}
```

The output window below the code editor shows the result of running the program: "9".

También se pueden usar “if/else” para verificar si una variable es “null” y manejarla adecuadamente. El operador Elvis (“?:”) permite proporcionar un valor predeterminado si la variable es “null”. Esto facilita el manejo seguro de valores nulos en el código.

The screenshot shows the Kotlin Play IDE interface. The code editor contains the following Kotlin code, which includes an if/else statement to handle the nullable string:

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    var favoriteActor: String? = "Sandra Oh"

    if (favoriteActor != null) {
        println("The number of characters in your favorite actor's name is ${favoriteActor.length}.")
    }
}
```

The output window below the code editor shows the result of running the program: "The number of characters in your favorite actor's name is 9.".

The screenshot shows the Kotlin Play IDE interface. The code editor contains the following Kotlin code:

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    var favoriteActor: String? = null

    if(favoriteActor != null) {
        println("The number of characters in your favorite actor's name is ${favoriteActor.length}.")
    } else {
        println("You didn't input a name.")
    }
}
```

The output window below the code editor displays the message: "You didn't input a name." This indicates that the variable "favoriteActor" was not assigned a value before the program was run.

En el siguiente código Kotlin podemos observar que se comienza declarando una variable mutable y nullable llamada “favoriteActor”, a la cual se le asigna inicialmente la cadena “Sandra Oh”. Posteriormente, se declara una variable inmutable llamada “lengthOfName”, cuyo valor se determina mediante una expresión condicional “if-else”. Esta condición verifica si la variable “favoriteActor” no es nula; en caso afirmativo, se obtiene la longitud de la cadena almacenada en “favoriteActor”, mientras que, si es nula, se asigna el valor 0 a “lengthOfName”. Finalmente, se utiliza la función “println” para imprimir en la consola un mensaje que incorpora el valor de “lengthOfName” mediante la interpolación de cadenas, mostrando la cantidad de caracteres en el nombre del actor favorito.

The screenshot shows the Kotlin Play IDE interface. The code editor contains the following Kotlin code:

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    var favoriteActor: String? = "Sandra Oh"

    val lengthOfName = if (favoriteActor != null) {
        favoriteActor.length
    } else {
        0
    }

    println("The number of characters in your favorite actor's name is $lengthOfName.")
}
```

The output window below the code editor displays the message: "The number of characters in your favorite actor's name is 9.". This indicates that the variable "favoriteActor" was assigned the value "Sandra Oh" before the program was run, and the code successfully calculated its length.

Por último, en la función principal (main) se comienza con la declaración de una variable mutable y que puede contener valores nulos, “favoriteActor”, a la cual se le asigna la cadena "Sandra Oh". La siguiente línea calcula la longitud del nombre del actor favorito y la asigna a una variable inmutable llamada “lengthOfName”. Para manejar la posibilidad de que “favoriteActor” sea nulo, se emplea el operador elvis (“?:”). La expresión “favoriteActor?.length” realiza una llamada segura: si “favoriteActor” no es nulo, se accede a su propiedad length; de lo contrario, la expresión completa evalúa a null. El operador elvis luego entra en juego, de manera que si el resultado de la parte izquierda es null, se devuelve el valor de la derecha, que en este caso es 0. Finalmente, se imprime en la consola un mensaje que incluye la longitud del nombre, utilizando la interpolación de cadenas.

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    var favoriteActor: String? = "Sandra Oh"
    val lengthOfName = favoriteActor?.length ?: 0
    println("The number of characters in your favorite actor's name is $lengthOfName.")
}
```

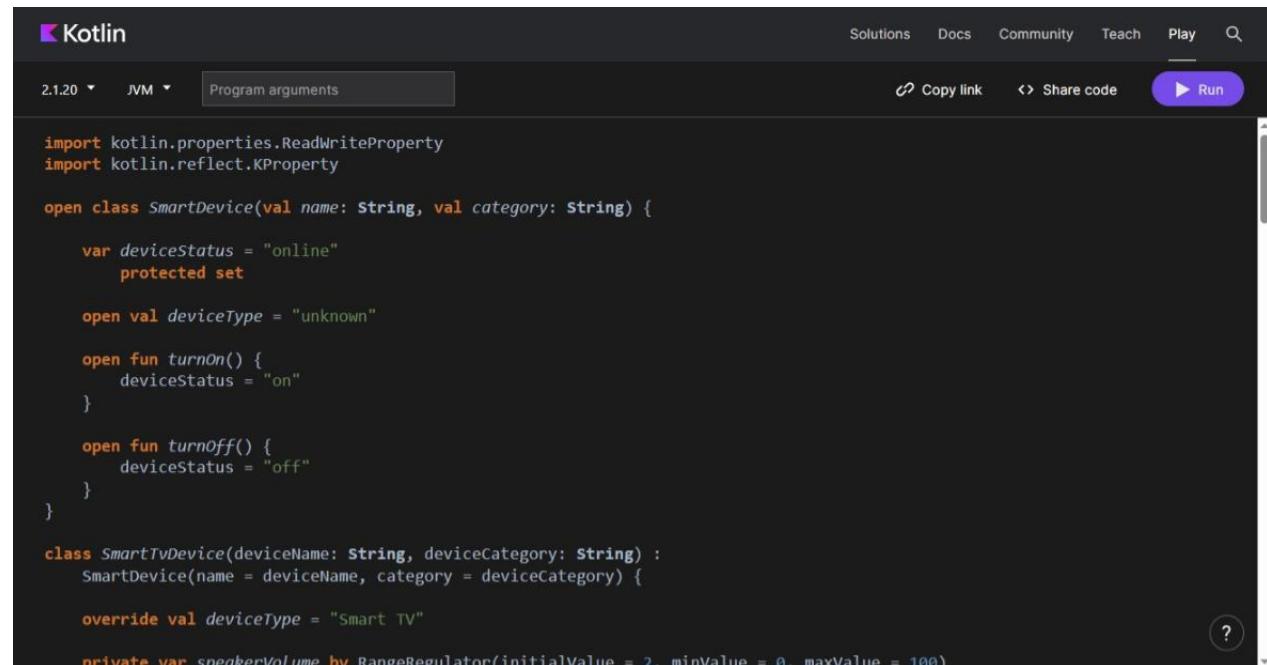
The number of characters in your favorite actor's name is 9.

## Usa clases y objetos en Kotlin

En Kotlin, las clases se definen con la palabra clave “class” y contienen propiedades y métodos. Los constructores se usan para crear instancias de objetos, y la herencia permite la reutilización de código (relación IS-A). La composición (relación HAS-A) se usa para combinar objetos dentro de otros objetos. Los modificadores de visibilidad (public, private, protected, internal) controlan el acceso a las propiedades y métodos. Además, los delegados de propiedades permiten reutilizar el código de los métodos “get” y “set”.

En este código Kotlin, se establece una jerarquía de dispositivos inteligentes comenzando con la clase base SmartDevice, la cual define las características fundamentales como name, category y deviceStatus, siendo este último modificable solo por la propia clase y sus herederas a través del protected set. Además, introduce una propiedad virtual deviceType con un valor predeterminado y declara funciones virtuales turnOn() y turnOff() para gestionar el estado del dispositivo.

La clase SmartTvDevice extiende SmartDevice para representar un televisor inteligente, sobrescribiendo deviceType a "Smart TV". Internamente, gestiona el speakerVolume y el channelNumber mediante la delegación de propiedades a la clase RangeRegulator, lo que asegura que estos valores se mantengan dentro de rangos específicos. Proporciona funciones para incrementar el volumen y cambiar de canal, y redefine los comportamientos de turnOn() y turnOff() para incluir acciones propias de un televisor, como informar del volumen y el canal al encenderse.



The screenshot shows the Kotlin playground interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the navigation bar, there are dropdown menus for '2.1.20' and 'JVM', and a 'Program arguments' input field. To the right of these are 'Copy link', 'Share code', and a purple 'Run' button. The main area contains the following Kotlin code:

```
import kotlin.properties.ReadWriteProperty
import kotlin.reflect.KProperty

open class SmartDevice(val name: String, val category: String) {
    var deviceStatus = "online"
        protected set

    open val deviceType = "unknown"

    open fun turnOn() {
        deviceStatus = "on"
    }

    open fun turnOff() {
        deviceStatus = "off"
    }
}

class SmartTvDevice(deviceName: String, deviceCategory: String) :
    SmartDevice(name = deviceName, category = deviceCategory) {

    override val deviceType = "Smart TV"

    private var speakerVolume by RangeRegulator(initialValue = 2, minValue = 0, maxValue = 100)
}
```

Ilustración 1

The screenshot shows the Kotlin playground interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the navigation bar, the version is set to 2.1.20 and the JVM target is selected. A "Program arguments" input field is present. On the right side, there are buttons for "Copy link", "Share code", and a prominent "Run" button. The main area contains the following Kotlin code:

```
private var channelNumber by RangeRegulator(initialValue = 1, minValue = 0, maxValue = 200)

fun increaseSpeakerVolume() {
    speakerVolume++
    println("Speaker volume increased to $speakerVolume.")
}

fun nextChannel() {
    channelNumber++
    println("Channel number increased to $channelNumber.")
}

override fun turnOn() {
    super.turnOn()
    println(
        "$name is turned on. Speaker volume is set to $speakerVolume and channel number is " +
        "set to $channelNumber."
    )
}

override fun turnOff() {
    super.turnOff()
    println("$name turned off")
}
```

### Ilustración 2

Por su parte, la clase SmartLightDevice, también heredando de SmartDevice, representa una luz inteligente y establece su deviceType como "Smart Light". Similar a la televisión, utiliza RangeRegulator para controlar el brightnessLevel dentro de un rango definido. Ofrece una función para aumentar el brillo y personaliza las funciones turnOn() y turnOff() para establecer niveles de brillo específicos al cambiar el estado de la luz.

The screenshot shows the Kotlin playground interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the navigation bar, there are dropdown menus for '2.1.20' and 'JVM', and a 'Program arguments' input field. To the right of these are 'Copy link', 'Share code', and a 'Run' button. The main area contains the following Kotlin code:

```
class SmartLightDevice(deviceName: String, deviceCategory: String) : SmartDevice(name = deviceName, category = deviceCategory) {
    override val deviceType = "Smart Light"

    private var brightnessLevel by RangeRegulator(initialValue = 0, minValue = 0, maxValue = 100)

    fun increaseBrightness() {
        brightnessLevel++
        println("Brightness increased to $brightnessLevel.")
    }

    override fun turnOn() {
        super.turnOn()
        brightnessLevel = 2
        println("$name turned on. The brightness level is $brightnessLevel.")
    }

    override fun turnOff() {
        super.turnOff()
        brightnessLevel = 0
        println("Smart Light turned off")
    }
}
```

### Ilustración 3

La clase SmartHome actúa como un sistema de control centralizado, manteniendo instancias de SmartTvDevice y SmartLightDevice. Lleva un registro de la cantidad de dispositivos encendidos y ofrece métodos para interactuar con cada dispositivo individualmente, como encender y apagar la televisión y la luz, ajustar el volumen y el brillo, y cambiar el canal. También incluye una función para apagar todos los dispositivos conectados.

The screenshot shows the Kotlin playground interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the navigation bar, there are dropdown menus for '2.1.20' and 'JVM'. A 'Program arguments' input field is present. On the right side, there are buttons for 'Copy link', 'Share code', and a purple 'Run' button. The main area contains the following Kotlin code:

```
class SmartHome(
    val smartTvDevice: SmartTvDevice,
    val smartLightDevice: SmartLightDevice
) {

    var deviceTurnOnCount = 0
        private set

    fun turnOnTv() {
        deviceTurnOnCount++
        smartTvDevice.turnOn()
    }

    fun turnOffTv() {
        deviceTurnOnCount--
        smartTvDevice.turnOff()
    }

    fun increaseTvVolume() {
        smartTvDevice.increaseSpeakerVolume()
    }

    fun changeTvChannelToNext() {
        smartTvDevice.nextChannel()
    }
}
```

#### Ilustración 4

Finalmente, la clase RangeRegulator implementa la interfaz ReadWriteProperty, actuando como un delegado de propiedad para gestionar valores dentro de un rango definido. Mantiene un valor interno (fieldData) y las funciones getValue() y setValue() definen cómo se accede y se modifica este valor, asegurando que cualquier nuevo valor propuesto se encuentre dentro de los límites minValue y maxValue antes de ser aceptado.

La función main() sirve como punto de entrada del programa, donde se crean instancias de SmartTvDevice y SmartLightDevice, asignándolas a una variable de tipo SmartDevice para demostrar polimorfismo, y se llama a la función turnOn() en cada una de ellas para iniciar su funcionamiento.

```
fun turnOnLight() {
    deviceTurnOnCount++
    smartLightDevice.turnOn()
}

fun turnOffLight() {
    deviceTurnOnCount--
    smartLightDevice.turnOff()
}

fun increaseLightBrightness() {
    smartLightDevice.increaseBrightness()
}

fun turnOffAllDevices() {
    turnOffTv()
    turnOffLight()
}
}

class RangeRegulator(
    initialValue: Int,
    private val minValue: Int,
    private val maxValue: Int
) : ReadWriteProperty<Any?, Int> {
```

Ilustración 5

```
var fieldData = initialValue

override fun getValue(thisRef: Any?, property: KProperty<*>): Int {
    return fieldData
}

override fun setValue(thisRef: Any?, property: KProperty<*>, value: Int) {
    if (value in minValue..maxValue) {
        fieldData = value
    }
}

fun main() {
    var smartDevice: SmartDevice = SmartTvDevice("Android TV", "Entertainment")
    smartDevice.turnOn()

    smartDevice = SmartLightDevice("Google Light", "Utility")
    smartDevice.turnOn()
}
```

Ilustración 6

The screenshot shows the Kotlin Play IDE interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the navigation bar, there are dropdown menus for '2.1.20' and 'JVM'. A 'Program arguments' input field contains the value 'value'. To the right of these are 'Copy link', 'Share code', and a 'Run' button. The main area displays a block of Kotlin code:

```
        fieldData = value
    }
}
}

fun main() {
    var smartDevice: SmartDevice = SmartTvDevice("Android TV", "Entertainment")
    smartDevice.turnOn()

    smartDevice = SmartLightDevice("Google Light", "Utility")
    smartDevice.turnOn()
}
```

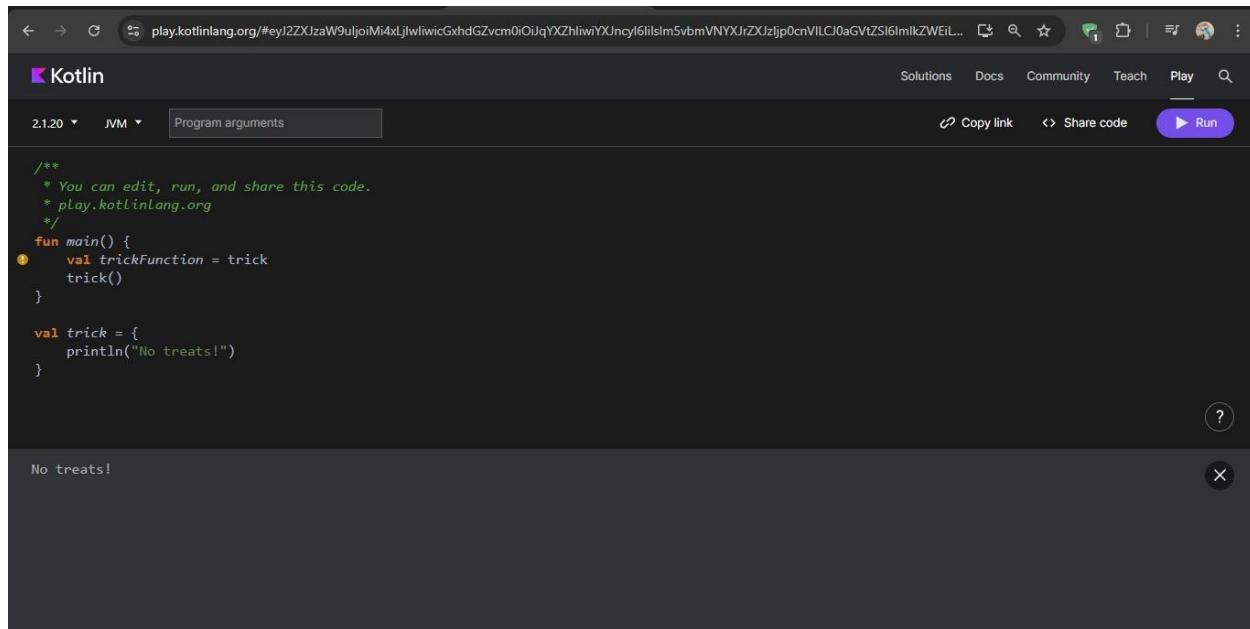
Below the code, the output window shows the results of the program execution:

```
Android TV is turned on. Speaker volume is set to 2 and channel number is set to 1.
Google Light turned on. The brightness level is 2.
```

Ilustración 7

## Cómo usar tipos de funciones y expresiones lambda en Kotlin

En Kotlin, las expresiones lambda ofrecen una sintaxis abreviada para definir funciones, y pueden ser pasadas como parámetros o devueltas desde otras funciones. Cuando se utiliza un solo parámetro en una lambda, se puede hacer referencia a él con “i”. Además, las expresiones lambda pueden ir al final de una llamada de función si el último parámetro es una función, usando una sintaxis más concisa. Mientras tanto, las funciones de orden superior son aquellas que toman otras funciones como parámetros o las devuelven. Un ejemplo es la función “repeat()”, que actúa como un bucle “for”.

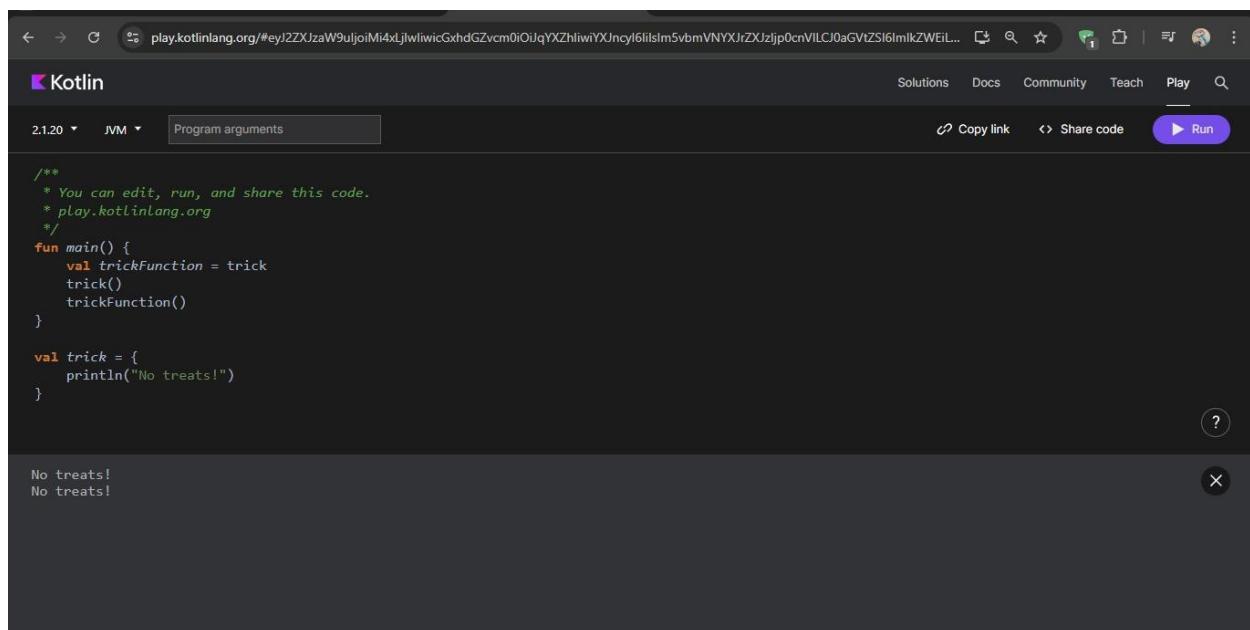


The screenshot shows a dark-themed web interface for running Kotlin code. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search bar. Below the bar, the main area has tabs for 2.1.20 and JVM, and a "Program arguments" input field. The code editor contains the following Kotlin code:

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    val trickFunction = trick
    trick()
}

val trick = {
    println("No treats!")
}
```

When the "Run" button is clicked, the output window shows the text "No treats!".

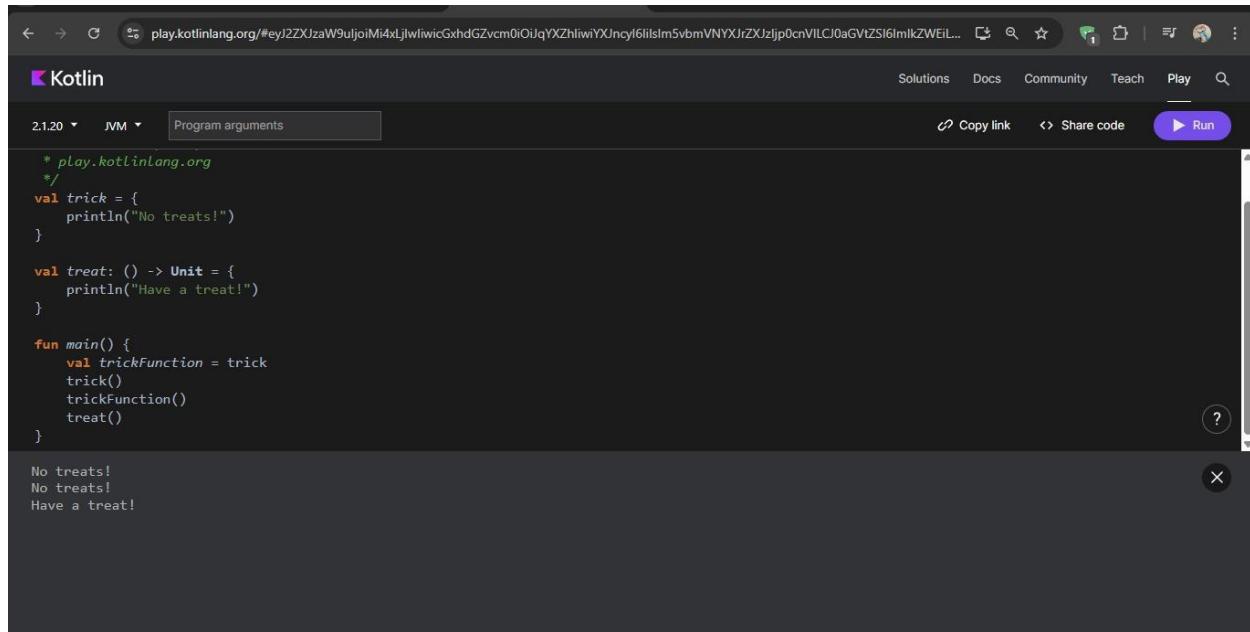


This screenshot shows a similar interface to the first one, but with more complex code. The code editor contains:

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    val trickFunction = trick
    trick()
    trickFunction()
}

val trick = {
    println("No treats!")
}
```

After running, the output window shows two lines: "No treats!" followed by "No treats!".



The screenshot shows the Kotlin Play IDE interface. The code editor contains the following Kotlin code:

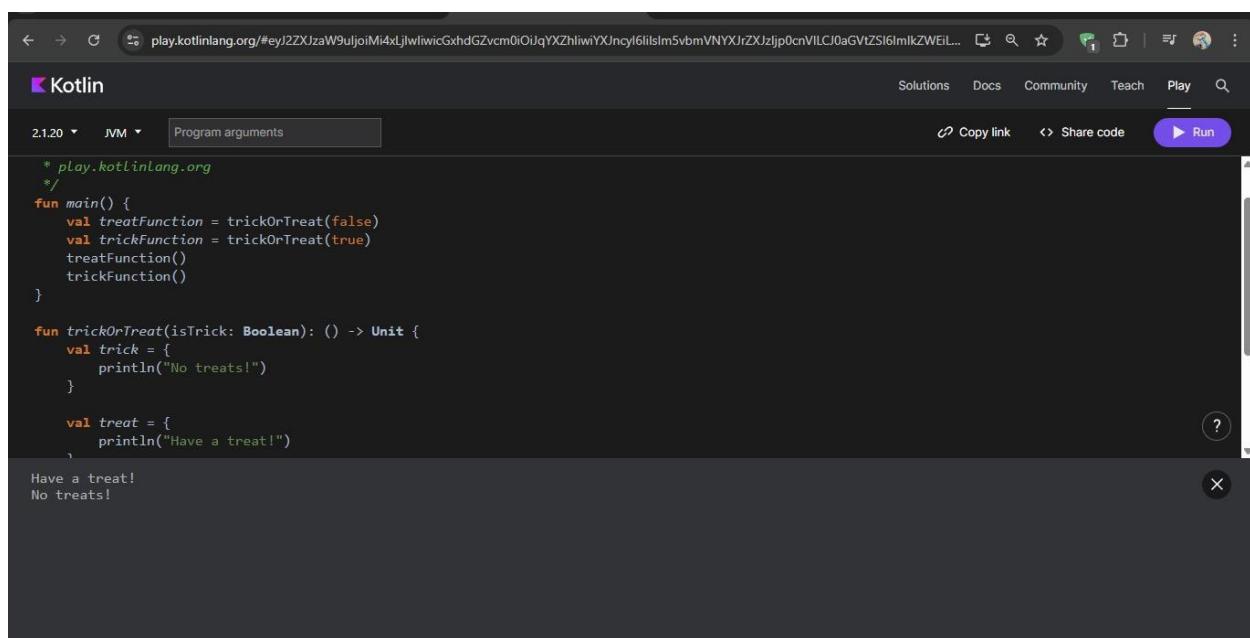
```
* play.kotlinlang.org
*/
val trick = {
    println("No treats!")
}

val treat: () -> Unit = {
    println("Have a treat!")
}

fun main() {
    val trickFunction = trick
    trick()
    trickFunction()
    treat()
}
```

The output window below the code shows the results of running the program:

```
No treats!
No treats!
Have a treat!
```



The screenshot shows the Kotlin Play IDE interface. The code editor contains the following Kotlin code:

```
* play.kotlinlang.org
*/
fun main() {
    val treatFunction = trickOrTreat(false)
    val trickFunction = trickOrTreat(true)
    treatFunction()
    trickFunction()
}

fun trickOrTreat(isTrick: Boolean): () -> Unit {
    val trick = {
        println("No treats!")
    }

    val treat = {
        println("Have a treat!")
    }
}
```

The output window below the code shows the results of running the program:

```
Have a treat!
No treats!
```

En el siguiente código podemos ver el concepto de funciones de orden superior y expresiones lambda. En la función main, se declaran dos variables que almacenan funciones: coins, que toma un entero y devuelve una cadena describiendo una cantidad de monedas, y cupcake, que devuelve una cadena indicando tener un pastelito. Luego, se llama a la función trickOrTreat, la cual decide si dar una "travesura" o un "trato" basándose en un valor booleano y una función adicional. En la primera llamada, con isTrick siendo false y pasando la función coins, trickOrTreat imprime el resultado de llamar a coins con el valor 5 ("5 quarters") y devuelve una función que imprime "Have a treat!". En la segunda llamada, con isTrick siendo true y pasando la función cupcake, trickOrTreat simplemente devuelve una función que

imprime "No treats!". Finalmente, se llama a las funciones devueltas por `trickOrTreat`, lo que resulta en la impresión de "Have a treat!" y luego "No treats!" en la consola.

The screenshot shows the Kotlin playground interface. The code defines a `main` function that prints "5 quarters", "Have a treat!", and "No treats!" sequentially. The output window shows the same three lines of text.

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    val coins: (Int) -> String = { quantity ->
        "$quantity quarters"
    }

    val cupcake: (Int) -> String = {
        "Have a cupcake!"
    }

    val treatFunction = trickOrTreat(false, coins)
    val trickFunction = trickOrTreat(true, cupcake)
    treatFunction()
}

5 quarters
Have a treat!
No treats!
```

The screenshot shows the Kotlin playground interface. The code is identical to the first one, but the `trickFunction` call is commented out with a `//`. The output window shows only the first two lines: "5 quarters" and "Have a treat!".

```
/*
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    val coins: (Int) -> String = { quantity ->
        "$quantity quarters"
    }

    val treatFunction = trickOrTreat(false, coins)
    val trickFunction = trickOrTreat(true, null)
    //treatFunction()
    //trickFunction()
}

5 quarters
Have a treat!
```

The screenshot shows the Kotlin Play IDE interface. The code editor contains the following Kotlin code:

```
/** * You can edit, run, and share this code. * play.kotlinlang.org */
fun main() {
    val coins: (Int) -> String = {
        "$it quarters"
    }

    val treatFunction = trickOrTreat(false, coins)
    val trickFunction = trickOrTreat(true, null)
    treatFunction()
    trickFunction()
}

fun trickOrTreat(isTrick: Boolean, extraTreat: ((Int) -> String)?): () -> Unit {
    5 quarters
    Have a treat!
    No treats!
}
```

The output window below the code editor shows the results of running the code:

```
5 quarters
Have a treat!
No treats!
```

The screenshot shows the Kotlin Play IDE interface. The code editor contains the following Kotlin code, which is identical to the one above but with a different lambda syntax:

```
/** * You can edit, run, and share this code. * play.kotlinlang.org */
fun main() {
    val treatFunction = trickOrTreat(false, { "$it quarters" })
    val trickFunction = trickOrTreat(true, null)
    treatFunction()
    trickFunction()
}

fun trickOrTreat(isTrick: Boolean, extraTreat: ((Int) -> String)?): () -> Unit {
    val trick = {
        println("No treats!")
    }
}

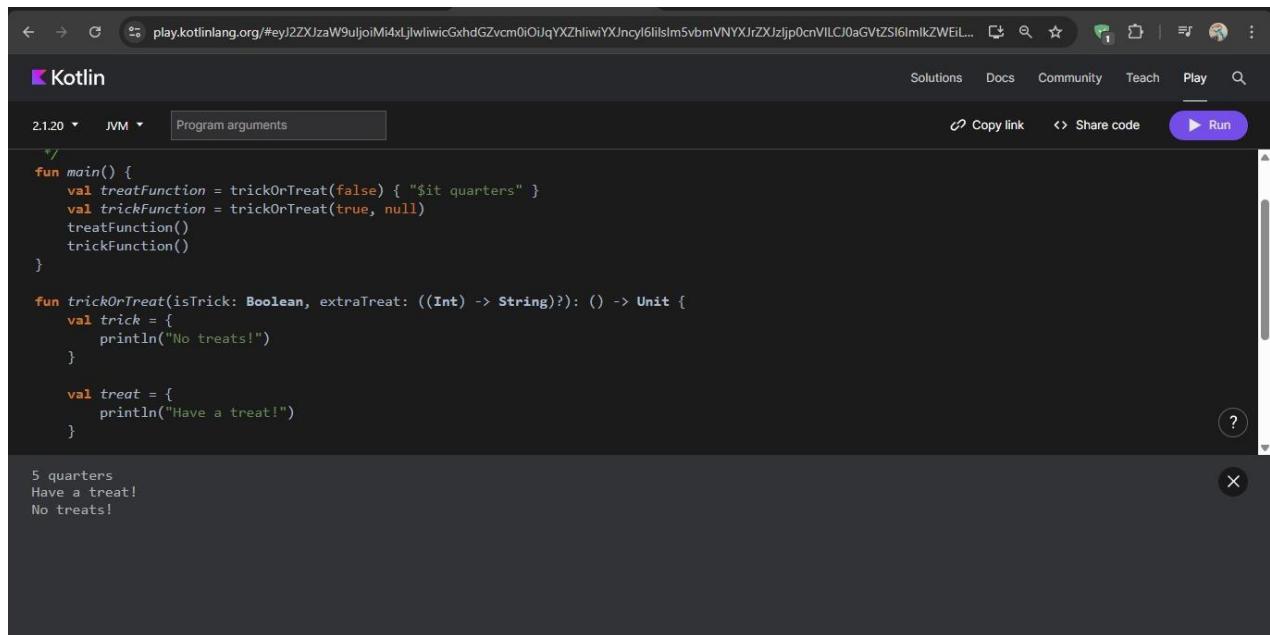
5 quarters
Have a treat!
No treats!
```

The output window below the code editor shows the results of running the code:

```
5 quarters
Have a treat!
No treats!
```

Por otra parte, en este código siguiente se ilustra una forma más elegante de pasar funciones como argumentos, especialmente cuando se trata de expresiones lambda al final de la lista de parámetros. En la función main, la primera llamada a trickOrTreat demuestra esta sintaxis: la lambda { "\$it quarters" } se coloca fuera de los paréntesis de la llamada, actuando como el último argumento. Esta lambda toma un entero y produce una cadena que describe una cantidad de monedas. La segunda llamada a trickOrTreat pasa null para el argumento extraTreat, que ahora es un tipo de función nullable. Dentro de la función trickOrTreat, si isTrick es falso, se verifica que extraTreat no sea nulo antes de invocarlo con el valor 5, lo que en el primer caso resulta en la impresión de "5 quarters". Finalmente, trickOrTreat

devuelve una función que imprime "Have a treat!" o "No treats!" dependiendo del valor de `isTrick`, y estas funciones devueltas son luego ejecutadas en `main`, produciendo la salida "Have a treat!" y "No treats!". El principal cambio aquí es la sintaxis concisa para la lambda final y el manejo explícito de un parámetro de función que puede ser nulo.



The screenshot shows the Kotlin Play IDE interface. The code editor contains the following Kotlin code:

```
/*
fun main() {
    val treatFunction = trickOrTreat(false) { "5 quarters" }
    val trickFunction = trickOrTreat(true, null)
    treatFunction()
    trickFunction()
}

fun trickOrTreat(isTrick: Boolean, extraTreat: ((Int) -> String)?): () -> Unit {
    val trick = {
        println("No treats!")
    }

    val treat = {
        println("Have a treat!")
    }
}
```

The output window below the code editor shows the execution results:

```
5 quarters
Have a treat!
No treats!
```

También se introduce la función `repeat()`, donde en la función `main`, se definen `treatFunction` y `trickFunction` de manera similar al ejemplo anterior, con `treatFunction` configurada para imprimir "Have a treat!" después de imprimir "5 quarters" debido a la llamada inicial a `trickOrTreat`, y `trickFunction` configurada para imprimir "No treats!". La novedad radica en el uso de `repeat(4) { treatFunction() }`, que ejecuta la función referenciada por `treatFunction` cuatro veces consecutivas, resultando en la impresión repetida del mensaje "Have a treat!". Finalmente, se llama a `trickFunction()`, lo que produce la impresión de "No treats!". La función `trickOrTreat` en sí mantiene su lógica previa para determinar qué función devolver basándose en la condición `isTrick` y la presencia de `extraTreat`.

The screenshot shows the Kotlin Play IDE interface. The code in the editor is:

```
2.1.20 JVM Program arguments
*/
fun main() {
    val treatFunction = trickOrTreat(false) { "it quarters" }
    val trickFunction = trickOrTreat(true, null)
    repeat(4) {
        treatFunction()
    }
    trickFunction()
}

fun trickOrTreat(isTrick: Boolean, extraTreat: ((Int) -> String)?): () -> Unit {
    val trick = {
        println("No treats!")
    }

    val treat = {
        5 quarters
        Have a treat!
        Have a treat!
        Have a treat!
        Have a treat!
        No treats!
    }
    if (isTrick) trick()
    else treat()
}
```

The output window shows the execution results:

```
5 quarters
Have a treat!
Have a treat!
Have a treat!
Have a treat!
No treats!
```

## Práctica: Conceptos básicos de Kotlin

Aquí se cubren conceptos fundamentales como sentencias condicionales (“if/else” y “when”), funciones de orden superior, expresiones lambda, y manejo de nulos con operadores seguros (“?.” y “?:”).

The screenshot shows the Kotlin Play IDE interface. The code in the editor is:

```
2.1.20 JVM Program arguments
fun main() {
    val morningNotification = 51
    val eveningNotification = 135

    printNotificationSummary(morningNotification)
    printNotificationSummary(eveningNotification)
}

fun printNotificationSummary(numberOfMessages: Int) {
    if (numberOfMessages < 100) {
        println("You have ${numberOfMessages} notifications.")
    } else {
        println("Your phone is blowing up! You have 99+ notifications.")
    }
}
```

The output window shows the execution results:

```
You have 51 notifications.
Your phone is blowing up! You have 99+ notifications.
```

Ilustración 8 Notificaciones Móviles

The screenshot shows the Kotlin playground interface. At the top, there are navigation links for Solutions, Docs, Community, Teach, Play, and a search icon. Below that, the version is set to 2.1.20 and the JVM target is selected. A "Program arguments" input field is present. On the right, there are "Copy link", "Share code", and "Run" buttons. The code in the editor is:

```
fun main() {
    val child = 5
    val adult = 28
    val senior = 87

    val isMonday = true

    println("The movie ticket price for a person aged $child is \$${ticketPrice(child, isMonday)}")
    println("The movie ticket price for a person aged $adult is \$${ticketPrice(adult, isMonday)}")
    println("The movie ticket price for a person aged $senior is \$${ticketPrice(senior, isMonday)}")
}

fun ticketPrice(age: Int, isMonday: Boolean): Int {
    return when(age) {
        in 0..12 -> 15
        in 13..60 -> if (isMonday) 25 else 30
        in 61..100 -> 20
        else -> -1
    }
}
```

The output window below shows the results of running the program:

```
The movie ticket price for a person aged 5 is $15.
The movie ticket price for a person aged 28 is $25.
The movie ticket price for a person aged 87 is $20.
```

Ilustración 9 Precios de Entradas de Cine

The screenshot shows the Kotlin playground interface. At the top, there are navigation links for Solutions, Docs, Community, Teach, Play, and a search icon. Below that, the version is set to 2.1.20 and the JVM target is selected. A "Program arguments" input field is present. On the right, there are "Copy link", "Share code", and "Run" buttons. The code in the editor is:

```
fun main() {
    printFinalTemperature(27.0, "Celsius", "Fahrenheit") { 9.0 / 5.0 * it + 32 }
    printFinalTemperature(350.0, "Kelvin", "Celsius") { it - 273.15 }
    printFinalTemperature(10.0, "Fahrenheit", "Kelvin") { 5.0 / 9.0 * (it - 32) + 273.15 }
}

fun printFinalTemperature(
    initialMeasurement: Double,
    initialUnit: String,
    finalUnit: String,
    conversionFormula: (Double) -> Double
) {
    val finalMeasurement = String.format("%.2f", conversionFormula(initialMeasurement)) // two decimal places
    println("$initialMeasurement degrees $initialUnit is $finalMeasurement degrees $finalUnit.")
}
```

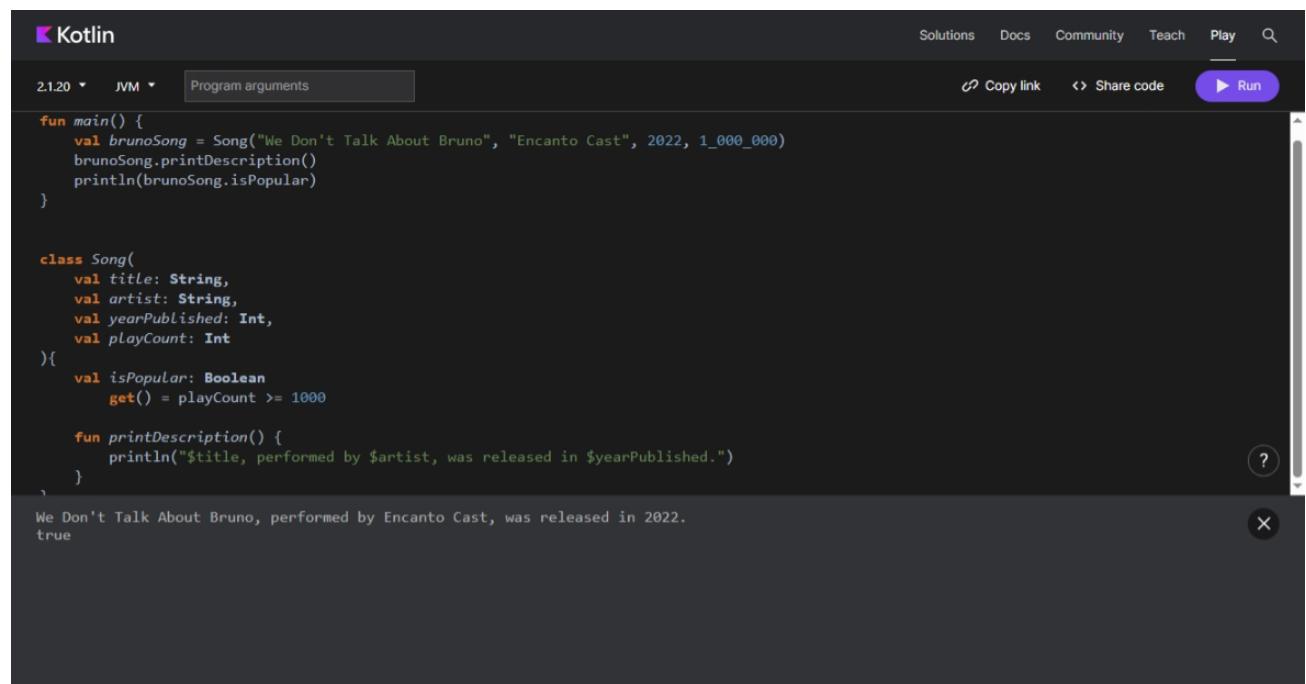
The output window below shows the results of running the program:

```
27.0 degrees Celsius is 80.60 degrees Fahrenheit.
350.0 degrees Kelvin is 76.85 degrees Celsius.
10.0 degrees Fahrenheit is 260.93 degrees Kelvin.
```

Ilustración 10 Conversor de Temperatura

Se enseña el uso de clases con propiedades personalizadas y métodos, como en la clase “Song”, y cómo manejar la herencia con clases como “Phone” y “FoldablePhone”.

En el código se incluye una clase llamada Song que demuestra la creación de un tipo de dato personalizado con sus propias propiedades (title, artist, yearPublished, playCount) y métodos (printDescription). Además, introduce el concepto de una propiedad calculada (isPopular), cuyo valor no se almacena directamente, sino que se calcula cada vez que se accede a ella, basándose en el valor de playCount. Esto ilustra cómo las clases en Kotlin pueden encapsular datos (propiedades) y comportamientos (métodos y propiedades calculadas) relacionados, permitiendo crear modelos más ricos y significativos para representar entidades del mundo real dentro del programa. La función main muestra cómo se crea una instancia de la clase Song y cómo se accede a sus propiedades y se llama a su método.



The screenshot shows the Kotlin Play IDE interface. The code editor contains a main function that creates a Song object and prints its description and popularity status. The output window at the bottom shows the printed results: "We Don't Talk About Bruno, performed by Encanto Cast, was released in 2022." followed by "true".

```
fun main() {
    val brunoSong = Song("We Don't Talk About Bruno", "Encanto Cast", 2022, 1_000_000)
    brunoSong.printDescription()
    println(brunoSong.isPopular)
}

class Song(
    val title: String,
    val artist: String,
    val yearPublished: Int,
    val playCount: Int
){
    val isPopular: Boolean
        get() = playCount >= 1000

    fun printDescription() {
        println("$title, performed by $artist, was released in $yearPublished.")
    }
}

We Don't Talk About Bruno, performed by Encanto Cast, was released in 2022.
true
```

Ilustración 11 Catalogo de Canciones

The screenshot shows the Kotlin playground interface. At the top, there are tabs for Solutions, Docs, Community, Teach, Play, and a search bar. Below the tabs, there are dropdown menus for version (2.1.20) and JVM. A "Program arguments" input field is present. On the right, there are buttons for Copy link, Share code, and Run. The main area contains the following Kotlin code:

```
fun main() {
    val amanda = Person("Amanda", 33, "play tennis", null)
    val atiqah = Person("Atiqah", 28, "climb", amanda)

    amanda.showProfile()
    atiqah.showProfile()
}

class Person(val name: String, val age: Int, val hobby: String?, val referrer: Person?) {
    fun showProfile() {
        println("Name: $name")
        println("Age: $age")
        if(hobby != null) {
            print("Likes to $hobby. ")
        }
        if(referrer != null) {
            print("Has a referrer named ${referrer.name}")
            if(referrer.hobby != null) {
                print(", who likes to ${referrer.hobby}.")
            } else {
                print(".")
            }
        } else {
            print("Doesn't have a referrer.")
        }
        print("\n\n")
    }
}
```

Ilustración 12 Perfil de Internet

The screenshot shows the Kotlin playground interface with the same layout as the previous one. The code is identical to the one in the first screenshot. A modal dialog box is displayed at the bottom, showing the output of the program's execution:

```
Name: Amanda
Age: 33
Likes to play tennis. Doesn't have a referrer.

Name: Atiqah
Age: 28
Likes to climb. Has a referrer named Amanda, who likes to play tennis.
```

Ilustración 12. 1 Perfil de Internet (Ejecución)

También se presenta un ejemplo de herencia con las clases Phone y FoldablePhone. La clase Phone actúa como una clase base (marcada como open para permitir la herencia), definiendo propiedades y funciones comunes a los teléfonos, como el estado de la pantalla (isScreenLightOn) y las funciones para encenderla (switchOn), apagarla (switchOff) y verificar su estado

(checkPhoneScreenLight). La clase FoldablePhone hereda de Phone, indicando una relación "es-un" (un FoldablePhone es un tipo de Phone). La clase FoldablePhone sobrescribe la función switchOn() para proporcionar una implementación específica para los teléfonos plegables: la pantalla solo se enciende si el teléfono no está plegado. Esto demuestra cómo la herencia permite crear jerarquías de clases donde las subclases pueden heredar características de la clase base y modificar o extender su comportamiento según sus necesidades particulares, promoviendo la reutilización de código y la creación de modelos más especializados. La función main (en el fragmento de Bid y auctionPrice) ilustraría cómo se pueden crear y utilizar instancias de estas clases.

The screenshot shows the Kotlin playground interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below that, a toolbar has dropdowns for '2.1.20' and 'JVM', and a 'Program arguments' input field. To the right of the input field are buttons for 'Copy link', 'Share code', and a purple 'Run' button. The main area contains the following Kotlin code:

```
open class Phone(var isScreenLightOn: Boolean = false){
    open fun switchOn() {
        isScreenLightOn = true
    }

    fun switchOff() {
        isScreenLightOn = false
    }

    fun checkPhoneScreenLight() {
        val phoneScreenLight = if (isScreenLightOn) "on" else "off"
        println("The phone screen's light is $phoneScreenLight.")
    }
}

class FoldablePhone(var isFolded: Boolean = true): Phone() {
    override fun switchOn() {
        if (!isFolded) {
            isScreenLightOn = true
        }
    }
}
```

Ilustración 13 Teléfonos Plegables

The screenshot shows the Kotlin playground interface. The code in the editor is:

```
fun fold() {
    isFolded = true
}

fun unfold() {
    isFolded = false
}

fun main() {
    val newFoldablePhone = FoldablePhone()

    newFoldablePhone.switchOn()
    newFoldablePhone.checkPhoneScreenLight()
    newFoldablePhone.unfold()
    newFoldablePhone.switchOn()
    newFoldablePhone.checkPhoneScreenLight()
}
```

The output window below the code shows two lines of text:

The phone screen's light is off.  
The phone screen's light is on.

Ilustración 13. 1 Teléfonos Plegables (Ejecución)

The screenshot shows the Kotlin playground interface. The code in the editor is:

```
fun main() {
    val winningBid = Bid(5000, "Private Collector")

    println("Item A is sold at ${auctionPrice(winningBid, 2000)}")
    println("Item B is sold at ${auctionPrice(null, 3000)}")
}

class Bid(val amount: Int, val bidder: String)

fun auctionPrice(bid: Bid?, minimumPrice: Int): Int {
    return bid?.amount ?: minimumPrice
}
```

The output window below the code shows two lines of text:

Item A is sold at 5000.  
Item B is sold at 3000.

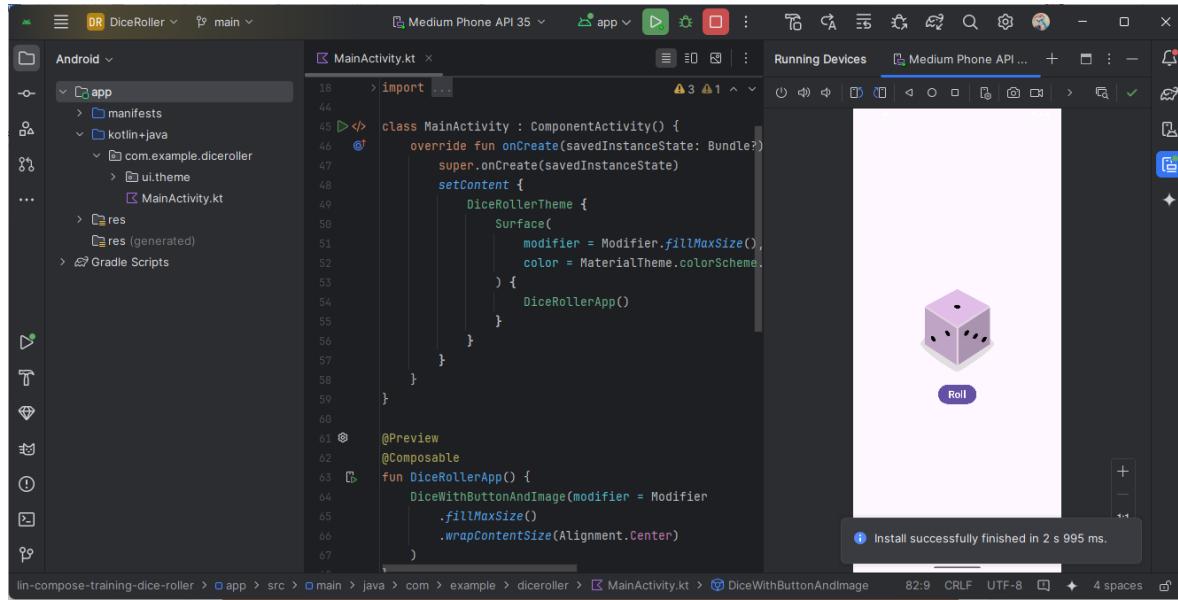
Ilustración 14 Subasta Especial

## Ruta de Aprendizaje 2 (Agrega un Botón a una app)

### Crear App interactiva de Dice Roller

En este codelab se crea una app interactiva de Dice Roller usando Jetpack Compose. Se definen funciones componibles, interfaces con composiciones, y

elementos como “Button” e “Image”. Además, se nos enseña a importar recursos gráficos, hacer la interfaz interactiva con “remember” y “mutableStateOf()” para almacenar y observar el estado, permitiendo que la IU se actualice dinámicamente al lanzar el dado.



En el archivo principal (MainActivity.kt), se define una función componible llamada DiceRollerApp(), que estructura la interfaz de la app. Dentro de ella, se utiliza remember { mutableStateOf(1) } para almacenar y observar el número del dado que ha salido, de forma que cada vez que cambia, Compose actualiza automáticamente la imagen mostrada. Se usan elementos componibles como Image para mostrar la cara del dado (imagen ubicada en res/drawable), y Button para permitir al usuario lanzar el dado. Al presionar el botón, se genera un número aleatorio entre 1 y 6 con Random.nextInt(1, 7), y se actualiza el estado.

### Use the debugger in Android Studio

En este codelab, aprendimos a usar el depurador en Android Studio para inspeccionar lo que sucede en la app de Dice Roller durante el tiempo de ejecución.

El depurador es una herramienta esencial que te permite inspeccionar la ejecución del código que potencia tu app para Android, de modo que puedas corregir cualquier error que aparezca. Te permite especificar puntos en los cuales suspender la ejecución del código e interactuar manualmente con variables, métodos y otros aspectos del código.

Existen dos maneras de ejecutar el depurador:

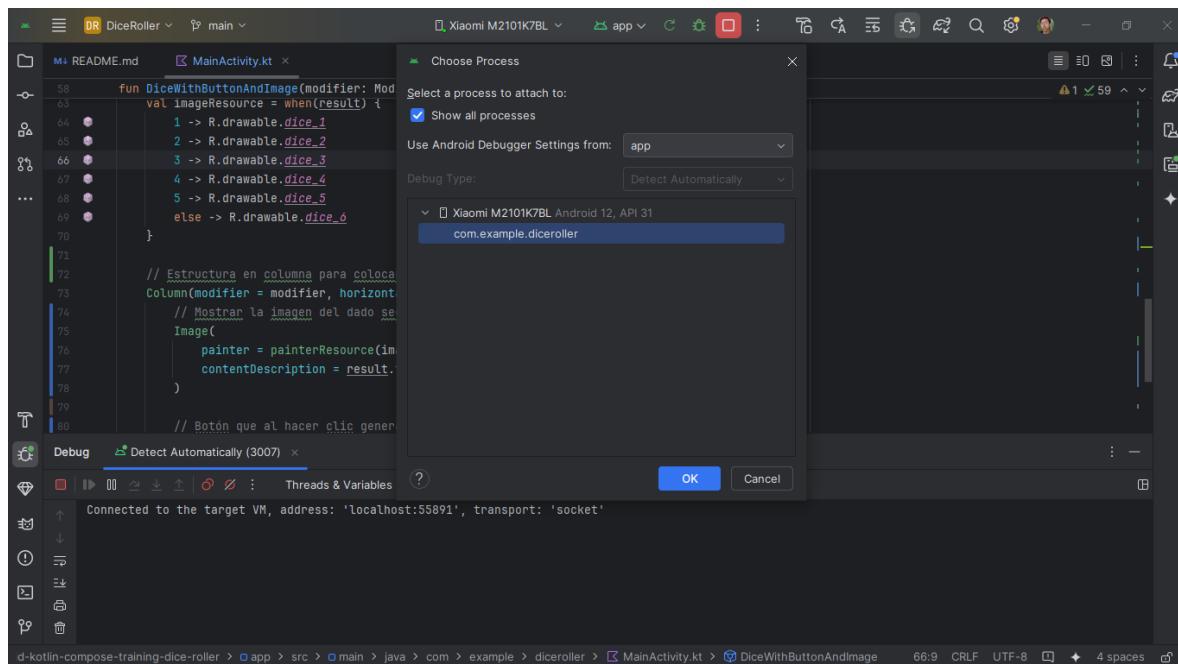
- ✓ Adjunta el depurador a un proceso de la app existente que se ejecuta en un dispositivo o emulador.
- ✓ Ejecuta la app con el depurador.

Ambos métodos logran lo mismo hasta cierto punto. Una vez que conozcas ambos, puedes elegir el que prefieras o el que sea necesario.

## Cómo adjuntar el depurador a un proceso de la aplicación

Para adjuntar el depurador a un proceso de la app, seguimos los siguientes pasos:

1. Haz clic en Attach debugger to Android process. Se abrirá el diálogo Choose Process, en el que podrás elegir el proceso al que deseas adjuntar el depurador.
2. Selecciona com.example.diceroller y haz clic en OK.

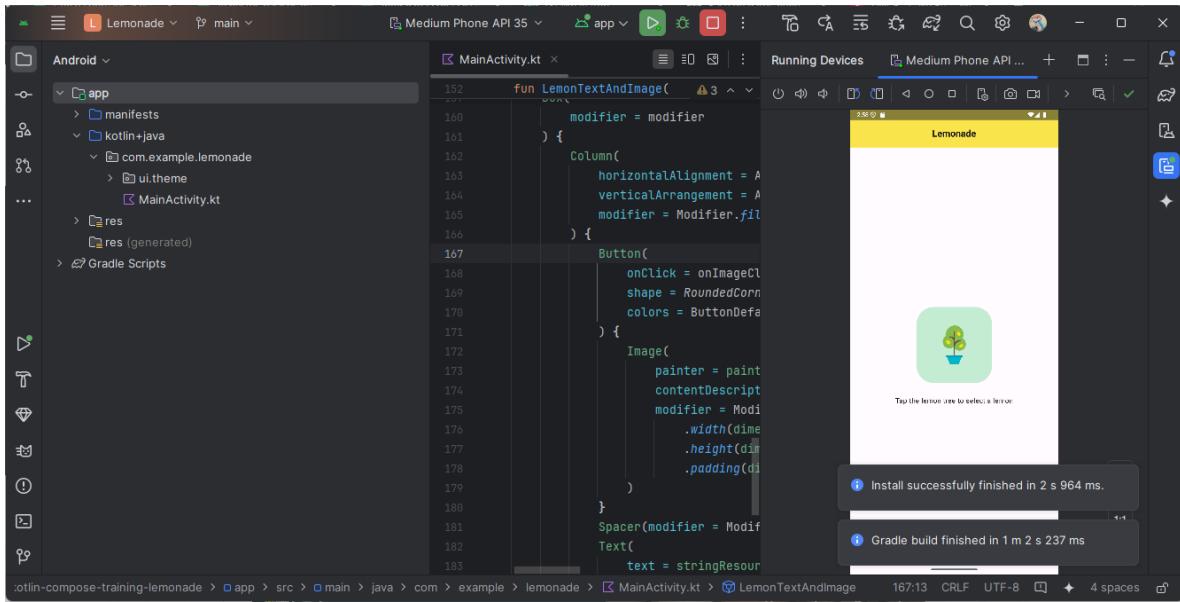


En la parte inferior de Android Studio, aparecerá un panel Debug con un mensaje que indica que el depurador está conectado al dispositivo de destino o al emulador.

Habremos adjuntado el depurador a la aplicación con éxito.

## Comportamiento de clics (Practice: Click behavior)

Este codelab nos enseña a construir una interfaz interactiva con Jetpack Compose guiando al usuario por cuatro pasos que simulan hacer limonada. Usa estado mutable ("remember" y "mutableStateOf") para cambiar imagen y texto según el paso: seleccionar un limón, exprimirlo varias veces, tomar limonada y reiniciar. Se usan componentes como "Column", "Image", "Text" y "Modifier.clickable" para crear una UI centrada y dinámica.



La app guía al usuario por cuatro etapas visuales: primero ve un limonero, luego un limón que debe exprimir (presionando varias veces), seguido de un vaso de limonada listo para beber, y finalmente un vaso vacío. Tras presionar el vaso vacío, el ciclo vuelve a comenzar.

El funcionamiento se basa en un estado mutable (`currentStep`) que determina qué imagen y texto mostrar. También se usa una variable adicional (`squeezeCount`) que se inicializa aleatoriamente entre 2 y 4 cuando se selecciona el limón, y decrece con cada clic hasta que el usuario puede avanzar al siguiente paso. La lógica está implementada usando `remember` y `mutableStateOf` para que la interfaz se actualice automáticamente al cambiar el estado.

Cada paso tiene una imagen (ubicada en los recursos `drawable`) y un texto específico, y toda la lógica está contenida en funciones composable como `LemonadeApp()` y `LemonadeScreen()`, donde se define el comportamiento interactivo y la disposición visual con `Column`, `Image`, y `Text`.

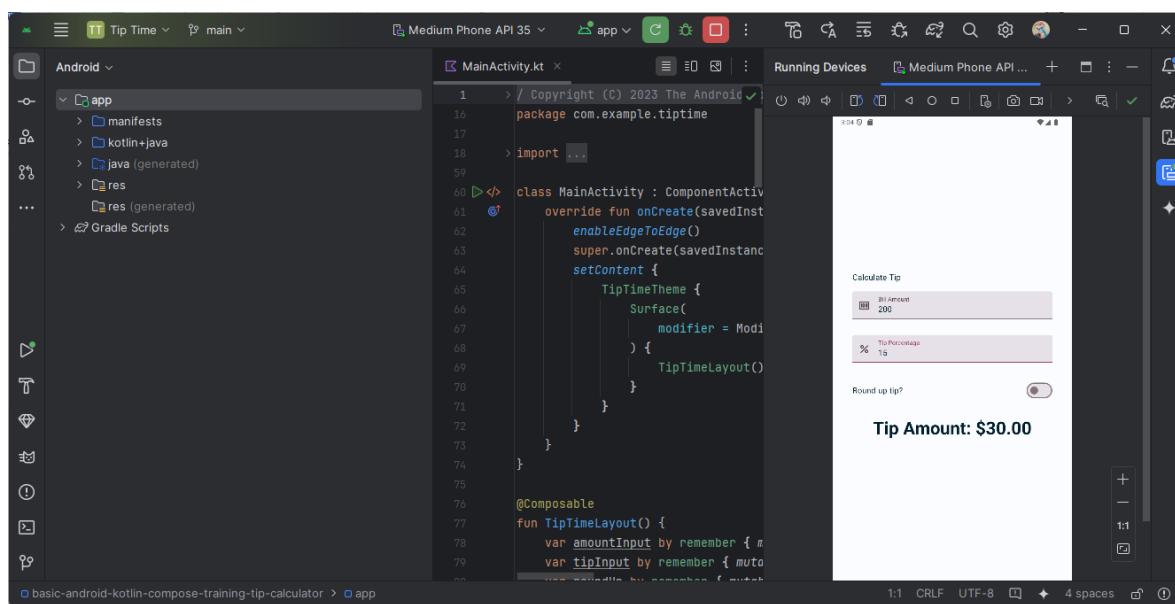
### **Ruta de Aprendizaje 3 (Interactúa con la IU y el estado)**

#### **Introducción al estado en Compose - Calcula una propina personalizada**

En este codelab se permite calcular propinas iniciando con una implementación básica donde se ingresa el monto de la factura mediante un campo de texto y la app calcula automáticamente una propina del 15%, mostrando el resultado en pantalla con un componente `Text`. Sin embargo, cabe mencionar que esta funcionalidad se extiende en una segunda lección para ofrecer una experiencia más completa en la cual se agrega un segundo campo de texto para que el usuario introduzca un porcentaje de propina personalizado, y un interruptor (`Switch`) que permite redondear el resultado al número entero más cercano. Todos estos componentes

se organizan dentro de una columna (Column) con desplazamiento vertical y alineaciones adecuadas, utilizando el sistema de diseño de Jetpack Compose.

El estado de cada campo (monto de la factura, porcentaje de propina y opción de redondeo) se gestiona con rememberSaveable para asegurar que los datos se conserven ante posibles recomposiciones o cambios de configuración. La función calculateTip() contiene la lógica de cálculo y aplica redondeo si se selecciona, devolviendo el resultado en formato de moneda mediante NumberFormat.getCurrencyInstance().format(). Para mejorar la experiencia del usuario, los campos de texto están optimizados para entrada numérica y etiquetados correctamente con stringResource, y los textos visibles se almacenan en strings.xml para facilitar la localización y el mantenimiento.



### Proyecto: Crea una app de Art Space

Creamos la aplicación para que sea de forma interactiva y dinámica para los usuarios, implementando dos botones para que los usuarios interactúen. Mostrando la obra de arte siguiente o anterior al presionar un botón.

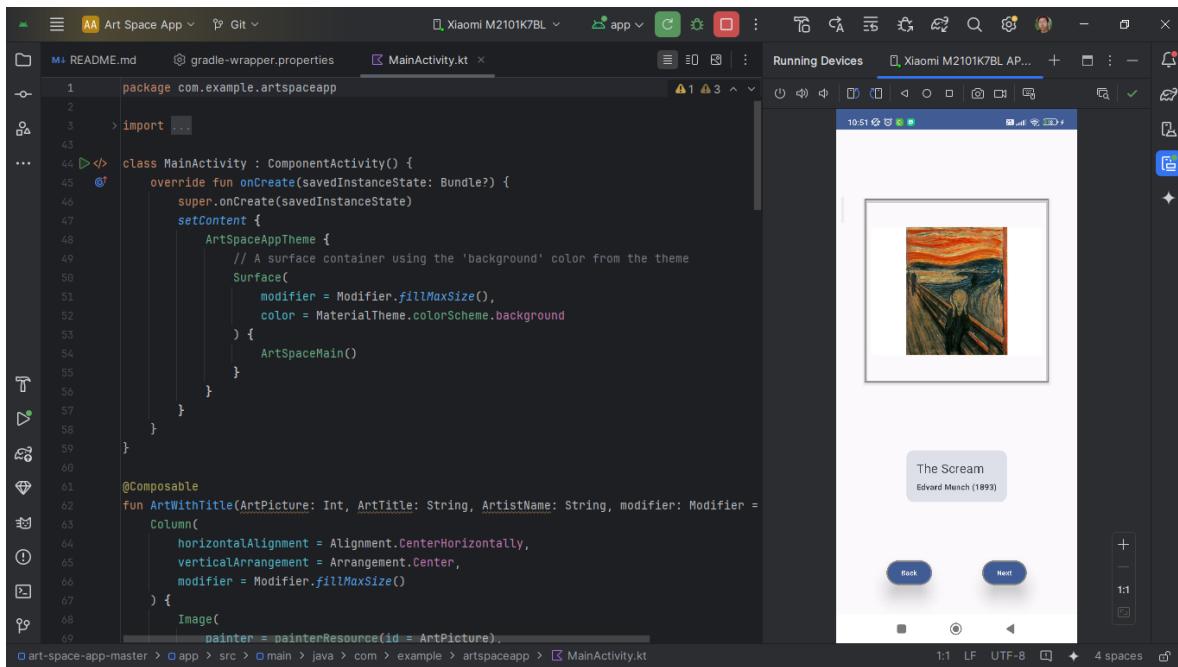
Trabajamos en la parte de la IU que muestra la obra de arte siguiente o anterior cuando se presiona un botón:

1. Primero, identificamos los elementos de la IU que deben cambiar según la interacción del usuario.

En este caso, los elementos de la IU son la imagen de la obra de arte, el título, el artista y el año.

2. Crea un estado para cada uno de los elementos dinámicos de la IU con el objeto MutableState.
3. Reemplazamos los valores codificados por states definidos.

Cuando los usuarios presionan el botón Next, deberían ver la siguiente obra de arte en la secuencia. Debido a que no tenemos una cantidad infinita de obras de arte, determinamos el comportamiento del botón Next para cuando se muestre la última obra de la serie para que vuelva a mostrar la primera obra de arte después de la última. Utilizando la sentencia when para compilar la lógica condicional, en lugar de las sentencias if else, para que el código sea más legible debido a la cantidad de casos de obras de arte.



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

- Project Structure:** Shows files like README.md, gradle-wrapper.properties, and MainActivity.kt.
- MainActivity.kt Code:**

```
1 package com.example.artspaceapp
2
3 import ...
4
5 class MainActivity : ComponentActivity() {
6     override fun onCreate(savedInstanceState: Bundle?) {
7         super.onCreate(savedInstanceState)
8         setContent {
9             ArtSpaceAppTheme {
10                 // A surface container using the 'background' color from the theme
11                 Surface(
12                     modifier = Modifier.fillMaxSize(),
13                     color = MaterialTheme.colorScheme.background
14                 ) {
15                     ArtSpaceMain()
16                 }
17             }
18         }
19     }
20 }
21
22 @Composable
23 fun ArtWithTitle(ArtPicture: Int, ArtTitle: String, ArtistName: String, modifier: Modifier =
24     Column(
25         horizontalAlignment = Alignment.CenterHorizontally,
26         verticalArrangement = Arrangement.Center,
27         modifier = Modifier.fillMaxSize()
28     ) {
29     Image(
30         painter = painterResource(id = ArtPicture),
31         contentDescription = ArtTitle
32     )
33     Text(
34         text = ArtistName
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
```
- Preview Window:** Displays the application's UI on an emulator. It shows a painting of "The Scream" by Edvard Munch. Below the image, a text box displays "The Scream" and "Edvard Munch (1893)". At the bottom, there are "Back" and "Next" buttons.
- Bottom Status Bar:** Shows device information (Xiaomi M2101K7BL), battery level (10:51), and connectivity status.

**Nota:** Los códigos con comentarios se encuentran subidos en [github](#).

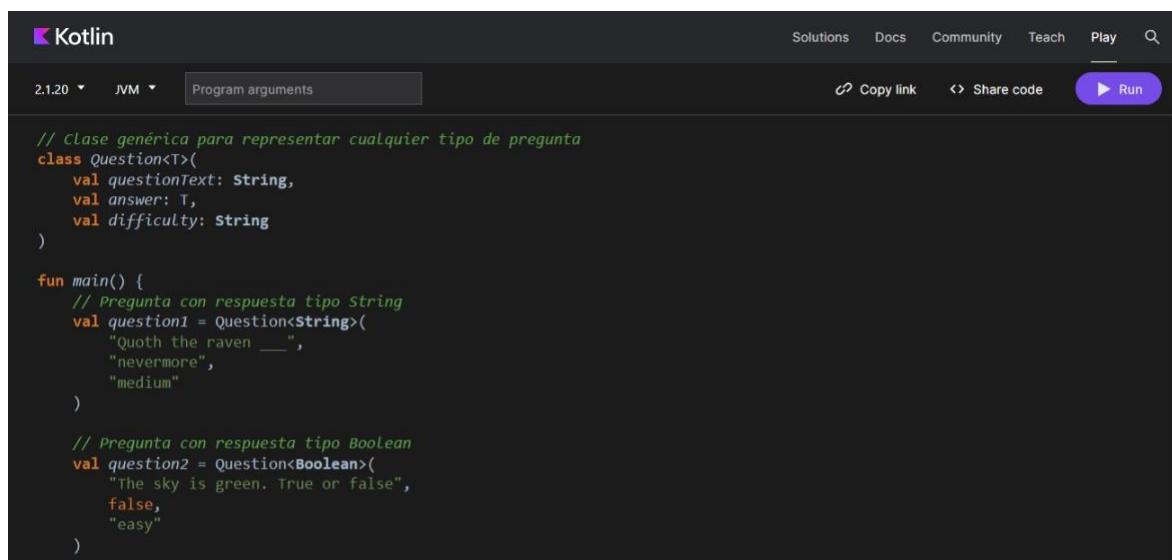
## Unidad 3: Cómo mostrar listas y usar Material Design

### Ruta de Aprendizaje 1 (Más aspectos básicos de Kotlin)

#### Parámetros genéricos, objetos y extensiones

En este codelab se abordan tres conceptos clave de Kotlin que son fundamentales para trabajar con Jetpack Compose y otros componentes de la interfaz de usuario. Los parámetros genéricos permiten crear clases y funciones reutilizables que pueden trabajar con diferentes tipos de datos, como listas o componentes de UI, sin necesidad de especificar un tipo concreto. Los objetos, por su parte, son instancias únicas de una clase que se utilizan para almacenar datos globales o funciones estáticas, como las instancias singleton. Finalmente, las extensiones permiten agregar nuevas funcionalidades a clases existentes sin modificar su código original, lo que facilita la personalización de los componentes de la UI en Jetpack Compose. Estos conceptos mejoran la flexibilidad, la reutilización y la modularidad del código, permitiendo crear aplicaciones más escalables y fáciles de mantener.

A continuación, podemos ver la creación y el uso de una clase genérica `Question<T>` para modelar preguntas de un cuestionario donde el tipo de la respuesta puede variar. En lugar de definir clases separadas para preguntas con respuestas de tipo String, Boolean, o Int, se utiliza una única clase genérica `Question` que utiliza un marcador de tipo `T` para la propiedad `answer`. Al crear instancias de la clase `Question` en la función `main()`, se especifica el tipo concreto para `T` (String para la primera pregunta, Boolean para la segunda e Int para la tercera), permitiendo que cada objeto `Question` almacene la respuesta con el tipo de dato adecuado. Finalmente, se imprime la información de cada pregunta.



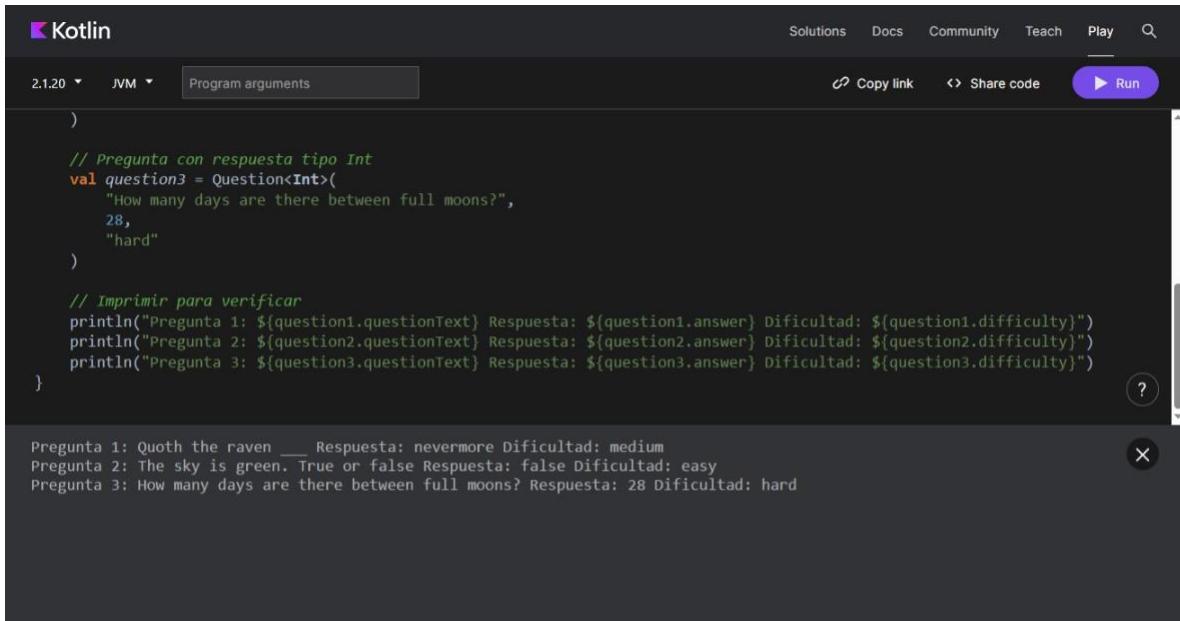
The screenshot shows the Kotlin playground interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the bar, there are dropdown menus for version (2.1.20) and JVM, and a "Program arguments" input field. On the right side, there are buttons for "Copy link", "Share code", and a large "Run" button. The main area contains the following Kotlin code:

```
// Clase genérica para representar cualquier tipo de pregunta
class Question<T>(
    val questionText: String,
    val answer: T,
    val difficulty: String
)

fun main() {
    // Pregunta con respuesta tipo String
    val question1 = Question<String>(
        "Quoth the raven ___",
        "nevermore",
        "medium"
    )

    // Pregunta con respuesta tipo Boolean
    val question2 = Question<Boolean>(
        "The sky is green. True or false",
        false,
        "easy"
    )
}
```

Imagen 1. Cómo crear una clase reutilizable con genéricos

A screenshot of the Kotlin playground interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the navigation bar, there are dropdown menus for version (2.1.20) and JVM, and a "Program arguments" input field. On the right side, there are buttons for "Copy link", "Share code", and a purple "Run" button. The main area contains Java code. The code defines a class Question<T> with properties questionText, answer, and difficulty. It also includes a main() function that creates three instances of Question<Int> and prints them to the console. The output window shows the printed results for each question: Pregunta 1, Pregunta 2, and Pregunta 3, each with its text, answer, and difficulty level.

```
// Pregunta con respuesta tipo Int
val question3 = Question<Int>(
    "How many days are there between full moons?",
    28,
    "hard"
)

// Imprimir para verificar
println("Pregunta 1: ${question1.questionText} Respuesta: ${question1.answer} Dificultad: ${question1.difficulty}")
println("Pregunta 2: ${question2.questionText} Respuesta: ${question2.answer} Dificultad: ${question2.difficulty}")
println("Pregunta 3: ${question3.questionText} Respuesta: ${question3.answer} Dificultad: ${question3.difficulty}")
}

Pregunta 1: Quoth the raven ____ Respuesta: nevermore Dificultad: medium
Pregunta 2: The sky is green. True or false Respuesta: false Dificultad: easy
Pregunta 3: How many days are there between full moons? Respuesta: 28 Dificultad: hard
```

Imagen 2. Cómo crear una clase reutilizable con genéricos (Ejecución)

También se representa un conjunto limitado de valores, en este caso, los niveles de dificultad de las preguntas de un cuestionario. Primero, se define una clase enum Difficulty con tres posibles valores: EASY, MEDIUM, y HARD. Luego, en la clase genérica Question<T>, el tipo de la propiedad difficulty se modifica de String a la nueva clase enum Difficulty. Finalmente, al crear instancias de Question en la función main(), se utilizan las constantes enum (como Difficulty.MEDIUM, Difficulty.EASY, y Difficulty.HARD) para asignar la dificultad a cada pregunta. Esto proporciona una forma más segura y legible de manejar los niveles de dificultad, evitando errores de escritura y facilitando futuras modificaciones.

The screenshot shows the Kotlin Play IDE interface. The code editor contains the following Kotlin code:

```
// Enum para representar niveles de dificultad
enum class Difficulty {
    EASY, MEDIUM, HARD
}

// Clase genérica para representar preguntas
class Question<T>(
    val questionText: String,
    val answer: T,
    val difficulty: Difficulty
)

fun main() {
    // Pregunta con respuesta tipo String
    val question1 = Question<String>(
        "Quoth the raven ___",
        "nevermore",
        Difficulty.MEDIUM
    )

    // Pregunta con respuesta tipo Boolean
    val question2 = Question<Boolean>(
        "The sky is green. True or false",
        false,
        Difficulty.EASY
    )
}
```

Imagen 3. Cómo usar una clase enum

The screenshot shows the Kotlin Play IDE interface after running the code. The terminal output window displays the following text:

```
Pregunta 1: Quoth the raven ___ Respuesta: nevermore Dificultad: MEDIUM
Pregunta 2: The sky is green. True or false Respuesta: false Dificultad: EASY
Pregunta 3: How many days are there between full moons? Respuesta: 28 Dificultad: HARD
```

Imagen 4. Cómo usar una clase enum (Ejecución)

A continuación, se muestra cómo al llamar a `toString()` en una clase regular (`Question` sin el prefijo `data`) donde la salida es simplemente el nombre de la clase y una referencia de memoria. Luego, se introduce la palabra clave `data` antes de la definición de la clase `Question`. Al convertir `Question` en una `data class`, el compilador de Kotlin genera automáticamente implementaciones para varios métodos útiles, incluyendo `equals()`, `hashCode()`, `toString()`, `componentN()` y `copy()`. La salida después de la modificación muestra que `toString()` ahora imprime una representación legible de las propiedades de la instancia de la clase `Question`.

The screenshot shows the Kotlin playground interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the navigation bar, the version is set to 2.1.20 and the JVM target is selected. A "Program arguments" input field is present. On the right side, there are "Copy link", "Share code", and "Run" buttons. The main area contains the following Kotlin code:

```
// Enum para representar niveles de dificultad
enum class Difficulty {
    EASY, MEDIUM, HARD
}

// Clase de datos genérica para representar preguntas
data class Question<T>(
    val questionText: String,
    val answer: T,
    val difficulty: Difficulty
)

fun main() {
    // Preguntas con diferentes tipos de respuesta
    val question1 = Question<String>(
        "Quoth the raven ___",
        "nevermore",
        Difficulty.MEDIUM
    )

    val question2 = Question<Boolean>(
        "The sky is green. True or false",
        false,
        Difficulty.EASY
    )
}
```

Imagen 5. Cómo usar una clase de datos

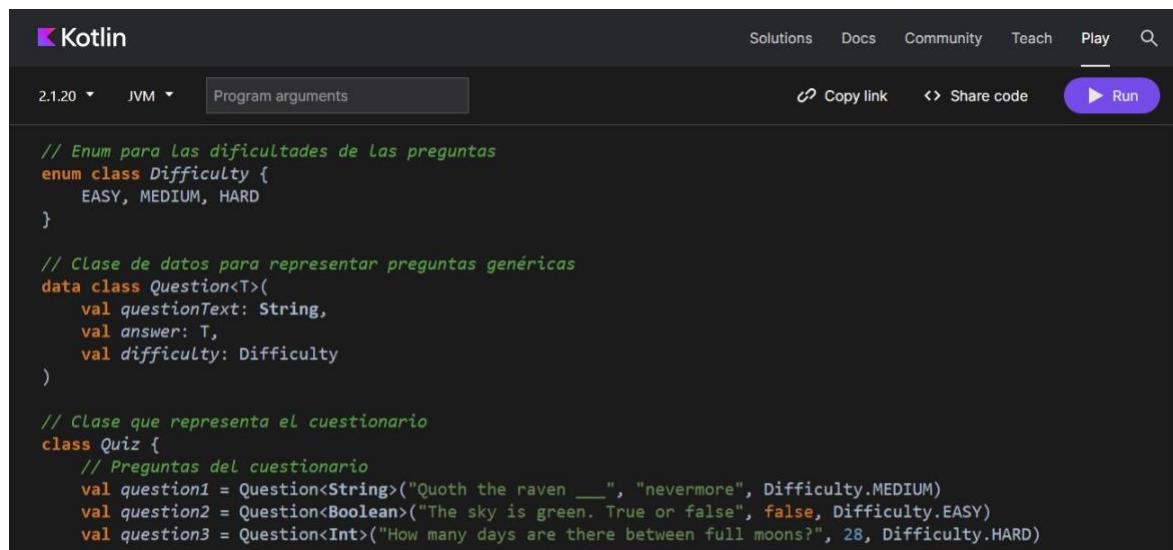
The screenshot shows the execution results of the code from Image 5. The output window displays the three created Question objects:

```
Question(questionText=Quoth the raven ___, answer=nevermore, difficulty=MEDIUM)
Question(questionText=The sky is green. True or false, answer=false, difficulty=EASY)
Question(questionText=How many days are there between full moons?, answer=28, difficulty=HARD)
```

Imagen 6. Cómo usar una clase de datos (Ejecución)

Se nos enseña también a crear objetos singleton usando la palabra clave `object`, asegurando que solo exista una instancia de la clase. Estos objetos se utilizan para gestionar estados únicos o servicios. Además, se pueden declarar objetos complementarios dentro de una clase usando `companion object`. Esto asocia el singleton con la clase, permitiendo acceder a sus propiedades directamente a través del nombre de la clase, como si fueran miembros estáticos. El siguiente ejemplo muestra cómo un `companion object` llamado `StudentProgress` dentro de la clase

Quiz rastrea el progreso del estudiante, accediéndose a sus propiedades total y answered mediante Quiz.total y Quiz.answered.

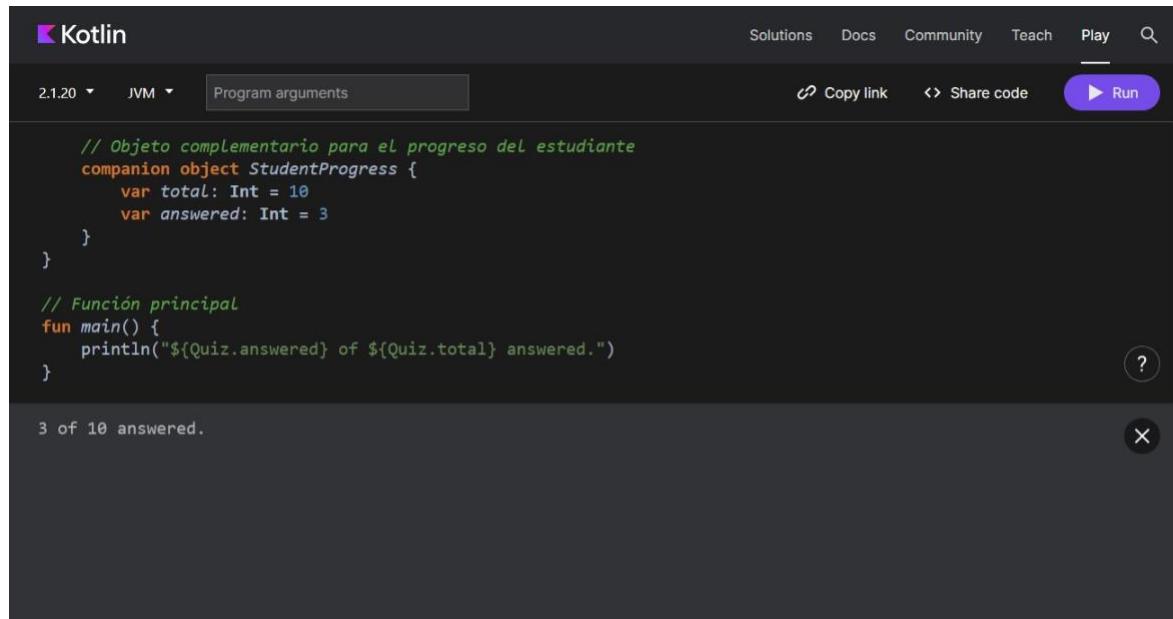


```
// Enum para las dificultades de las preguntas
enum class Difficulty {
    EASY, MEDIUM, HARD
}

// Clase de datos para representar preguntas genéricas
data class Question<T>(
    val questionText: String,
    val answer: T,
    val difficulty: Difficulty
)

// Clase que representa el cuestionario
class Quiz {
    // Preguntas del cuestionario
    val question1 = Question<String>("Quoth the raven ___", "nevermore", Difficulty.MEDIUM)
    val question2 = Question<Boolean>("The sky is green. True or false", false, Difficulty.EASY)
    val question3 = Question<Int>("How many days are there between full moons?", 28, Difficulty.HARD)
}
```

Imagen 7. Cómo usar un objeto singleton



```
// Objeto complementario para el progreso del estudiante
companion object StudentProgress {
    var total: Int = 10
    var answered: Int = 3
}

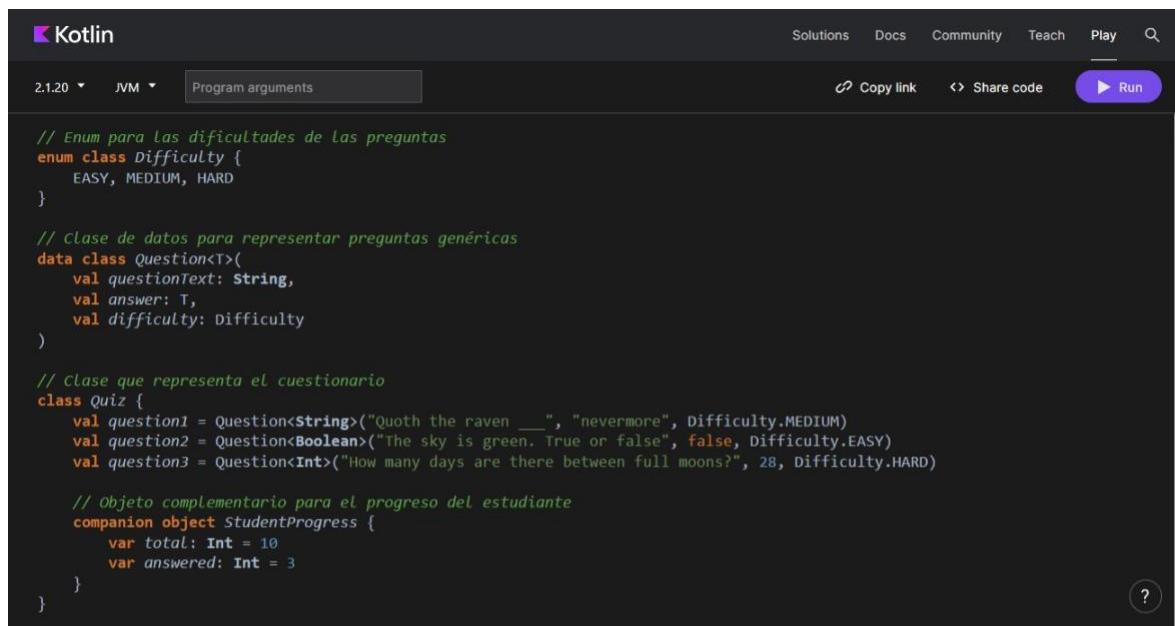
// Función principal
fun main() {
    println("${Quiz.answered} of ${Quiz.total} answered.")
}
```

3 of 10 answered.

Imagen 8. Cómo usar un objeto singleton (Ejecución)

También se nos muestra cómo se permite extender clases agregando propiedades y funciones de extensión sin modificar el código original. Se define una propiedad de extensión especificando el tipo a extender seguido de un punto y el nombre de la propiedad (con un getter). De manera similar, una función de extensión se define especificando el tipo a extender seguido de un punto y el nombre de la función. Esto permite usar la notación de punto para acceder a estas nuevas funcionalidades

como si fueran miembros de la clase extendida, facilitando la lectura y la extensibilidad del código.



The screenshot shows the Kotlin playground interface with the following code:

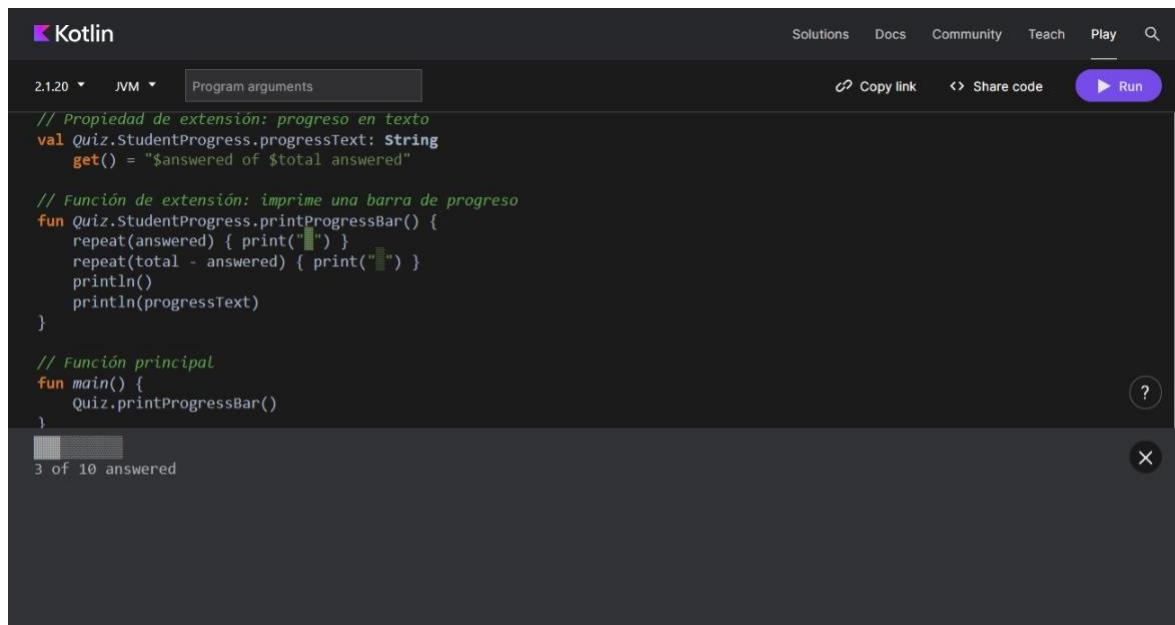
```
// Enum para las dificultades de las preguntas
enum class Difficulty {
    EASY, MEDIUM, HARD
}

// Clase de datos para representar preguntas genéricas
data class Question<T>(
    val questionText: String,
    val answer: T,
    val difficulty: Difficulty
)

// Clase que representa el cuestionario
class Quiz {
    val question1 = Question<String>("Quoth the raven __", "nevermore", Difficulty.MEDIUM)
    val question2 = Question<Boolean>("The sky is green. True or false", false, Difficulty.EASY)
    val question3 = Question<Int>("How many days are there between full moons?", 28, Difficulty.HARD)
}

// Objeto complementario para el progreso del estudiante
companion object StudentProgress {
    var total: Int = 10
    var answered: Int = 3
}
```

Imagen 9. Cómo extender clases con propiedades y métodos nuevos



The screenshot shows the Kotlin playground interface with the following code and its execution output:

```
// Propiedad de extensión: progreso en texto
val Quiz.StudentProgress.progressText: String
    get() = "$answered of $total answered"

// Función de extensión: imprime una barra de progreso
fun Quiz.StudentProgress.printProgressBar() {
    repeat(answered) { print("█") }
    repeat(total - answered) { print("█") }
    println()
    println(progressText)
}

// Función principal
fun main() {
    Quiz.printProgressBar()
}
```

Execution output:

```
3 of 10 answered
```

Imagen 10. Cómo extender clases con propiedades y métodos nuevos (Ejecución)

Kotlin permite definir interfaces (interface) que especifican propiedades y funciones que las clases deben implementar (:) y eso se muestra en el codelab. Se crea un contrato asegurando que las clases que implementan la interfaz proporcionen las funcionalidades definidas. En el ejemplo, se crea una interfaz ProgressPrintable con progressText y printProgressBar(). La clase Quiz implementa esta interfaz,

proporcionando su propia lógica para mostrar el progreso. En main(), se crea una instancia de Quiz para usar su printProgressBar(). Las interfaces ofrecen una forma estructurada y modular de compartir funcionalidades entre clases, a diferencia de las extensiones que agregan funcionalidades externamente.

```

Kotlin
2.1.20 JVM Program arguments Solutions Docs Community Teach Play Copy link Share code Run

// Enum para la dificultad de las preguntas
enum class Difficulty {
    EASY, MEDIUM, HARD
}

// Clase genérica para representar preguntas
data class Question<T>(
    val questionText: String,
    val answer: T,
    val difficulty: Difficulty
)

// INTERFAZ para imprimir progreso
interface ProgressPrintable {
    val progressText: String
    fun printProgressBar()
}

// Clase Quiz que ahora IMPLEMENTA ProgressPrintable
class Quiz : ProgressPrintable {

    val question1 = Question<String>("Quoth the raven __", "nevermore", Difficulty.MEDIUM)
    val question2 = Question<Boolean>("The sky is green. True or false", false, Difficulty.EASY)
    val question3 = Question<Int>("How many days are there between full moons?", 28, Difficulty.HARD)

    // Objeto complementario que lleva el progreso
    companion object StudentProgress {
        var total: Int = 10
        var answered: Int = 3
    }
}

```

Imagen 11. Cómo reescribir las funciones de extensión con interfaces

```

Kotlin
2.1.20 JVM Program arguments Solutions Docs Community Teach Play Copy link Share code Run

// Implementación de la propiedad progressText (obligatoria por la interfaz)
override val progressText: String
    get() = "${answered} of ${total} answered"

// Implementación de la función printProgressBar() (también requerida)
override fun printProgressBar() {
    repeat(answered) { print("#") }
    repeat(total - answered) { print("_") }
    println()
    println(progressText)
}

// Función principal
fun main() {
    Quiz().printProgressBar()
}

3 of 10 answered

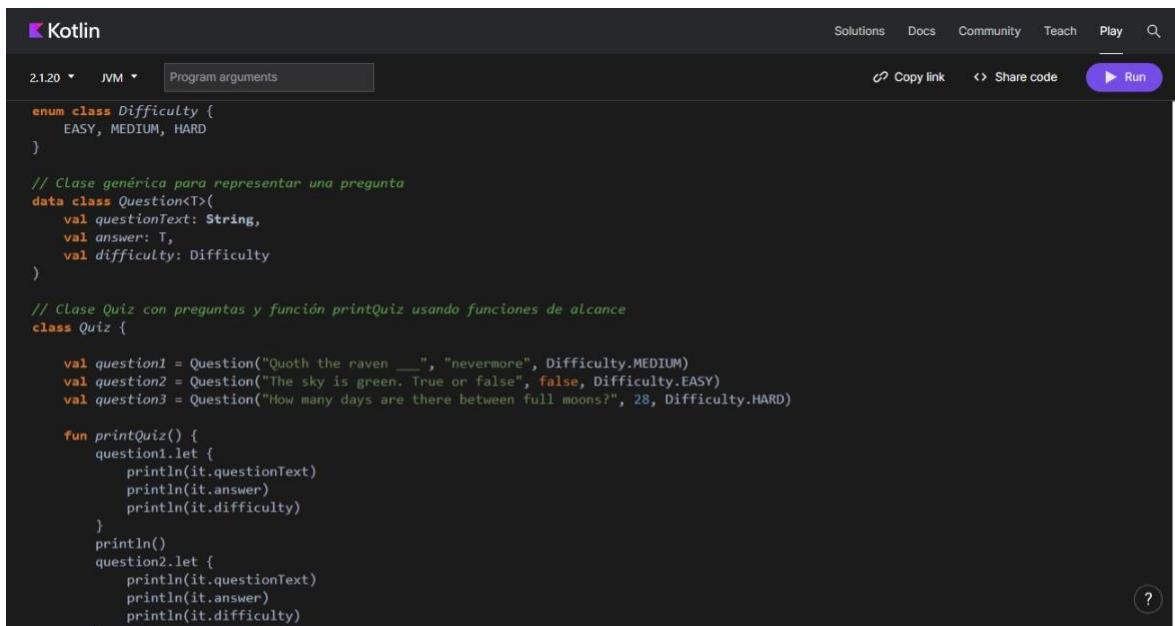
```

Imagen 12. Cómo reescribir las funciones de extensión con interfaces (Ejecución)

Ya por último se nos habla acerca de las funciones de alcance en Kotlin que permiten acceder a propiedades y métodos de un objeto de forma concisa dentro de una lambda, sin repetir el nombre del objeto.

Se explica la función let(), que permite referirse al objeto dentro de la lambda usando el identificador it. En el ejemplo, se agrega una función printQuiz() a la clase Quiz que originalmente accedía a las propiedades de question1, question2 y question3 usando sus nombres completos. Luego, se refactoriza esta función para usar let() en cada pregunta, reemplazando el nombre de la variable por it dentro de la lambda.

También se explica la función apply(), que permite llamar métodos en un objeto incluso antes de asignarlo a una variable. La lambda dentro de apply() tiene el objeto como receptor (this), aunque en este ejemplo no se usa explícitamente. Se muestra cómo crear una instancia de Quiz y llamar a printQuiz() dentro de un bloque apply(), incluso omitiendo la creación de una variable para la instancia de Quiz.



The screenshot shows the Kotlin playground interface with the following code:

```
Kotlin
2.1.20 • JVM • Program arguments
Solutions Docs Community Teach Play ⚙️ Copy link ⚙️ Share code Run

enum class Difficulty {
    EASY, MEDIUM, HARD
}

// Clase genérica para representar una pregunta
data class QuestionT<T>{
    val questionText: String,
    val answer: T,
    val difficulty: Difficulty
}

// Clase Quiz con preguntas y función printQuiz usando funciones de alcance
class Quiz {

    val question1 = Question("Quoth the raven ___", "nevermore", Difficulty.MEDIUM)
    val question2 = Question("The sky is green. True or false", false, Difficulty.EASY)
    val question3 = Question("How many days are there between full moons?", 28, Difficulty.HARD)

    fun printQuiz() {
        question1.let {
            println(it.questionText)
            println(it.answer)
            println(it.difficulty)
        }
        println()
        question2.let {
            println(it.questionText)
            println(it.answer)
            println(it.difficulty)
        }
    }
}
```

Imagen 13. Cómo usar funciones de alcance para acceder a las propiedades y los métodos de la clase

The screenshot shows the Kotlin Play IDE interface. At the top, there are tabs for Solutions, Docs, Community, Teach, Play, and a search bar. Below the tabs, there are dropdown menus for version (2.1.20) and JVM. A "Program arguments" input field is present. On the right, there are buttons for Copy link, Share code, and Run. The main area displays the following code and its output:

```
    println(it.answer)
    println(it.difficulty)
}
println()
question3.let {
    println(it.questionText)
    println(it.answer)
    println(it.difficulty)
}
println()
}

// Función main usando apply() para evitar declarar la variable
fun main() {
    Quiz().apply {
        printQuiz()
    }
}
Quoth the raven —
nevermore
MEDIUM

The sky is green. True or false
false
EASY

How many days are there between full moons?
28
HARD
```

Imagen 14. Cómo usar funciones de alcance para acceder a las propiedades y los métodos de la clase (Ejecución)

### Usa colecciones en Kotlin

Se resumen las características principales de los arrays, que almacenan datos ordenados del mismo tipo con un tamaño fijo, y cómo sirven de base para otras colecciones. Las listas se describen como colecciones ordenadas de tamaño variable, mientras que los conjuntos son colecciones no ordenadas que no admiten duplicados. Finalmente, los mapas se presentan como estructuras que almacenan pares de clave-valor, similares a los conjuntos en la unicidad de sus claves.

The screenshot shows the Kotlin Play IDE interface. At the top, there are tabs for Solutions, Docs, Community, Teach, Play, and a search bar. Below the tabs, there are dropdown menus for version (2.1.20) and JVM. A "Program arguments" input field is present. On the right, there are buttons for Copy link, Share code, and Run. The main area displays the following code and its output:

```
/*
 * Unidad 3
 */
fun main() {
    // Creamos un array de planetas rocosos
    val rockPlanets = arrayOf("Mercury", "Venus", "Earth", "Mars")

    // Creamos un array de planetas gaseosos (sin especificar tipo explícito)
    val gasPlanets = arrayOf("Jupiter", "Saturn", "Uranus", "Neptune")

    // Unimos los dos arrays en uno nuevo llamado solarSystem
    val solarSystem = rockPlanets + gasPlanets
}
```

The screenshot shows a Kotlin code editor window. The code is as follows:

```
/*
 * Unidad 3
 */
fun main() {
    // Creamos arrays como listas mutables
    val solarSystem = mutableListOf("Mercury", "Venus", "Earth", "Mars",
        "Jupiter", "Saturn", "Uranus", "Neptune")

    // Imprimimos todos los elementos
    println(solarSystem[0])
    println(solarSystem[1])
    println(solarSystem[2])
    println(solarSystem[3])
    println(solarSystem[4])
    println(solarSystem[5])
    println(solarSystem[6])
    println(solarSystem[7])
}
```

The output window below the code shows the following text:

```
Mercury
Venus
Earth
Mars
Jupiter
Saturn
Uranus
Neptune
Little Earth
Pluto
```

En este ejemplo que se muestra a continuación, dentro de la función main, se declara una variable inmutable llamada solarSystem a la cual se le asigna una lista inmutable que contiene los nombres de los ocho planetas del sistema solar, creada mediante la función listOf(). Posteriormente, se imprime en la consola el tamaño de esta lista utilizando la propiedad size. La salida del programa muestra el número 8, lo que indica que la lista solarSystem contiene ocho elementos, correspondientes a los ocho planetas definidos al crear la lista inmutable.

The screenshot shows a Kotlin code editor window. The code is as follows:

```
/*
 * Unidad 3
 */
fun main() {
    // Creamos una lista inmutable con los 8 planetas
    val solarSystem = listOf("Mercury", "Venus", "Earth", "Mars",
        "Jupiter", "Saturn", "Uranus", "Neptune")

    // Imprimimos el tamaño de la lista
    println(solarSystem.size)
}
```

The output window below the code shows the following text:

```
8
```

Kotlin

2.1.20 ▾ JVM ▾ Program arguments

Solutions Docs Community Teach Play

Copy link Share code Run

```
/*
 * Unidad 3
 */
fun main() {
    // Creamos una Lista inmutable con los 8 planetas
    val solarSystem = listOf("Mercury", "Venus", "Earth", "Mars",
        "Jupiter", "Saturn", "Uranus", "Neptune")

    // Accedemos al elemento en el índice 2 usando subíndice
    println(solarSystem[2]) // Debería imprimir: Earth

    // Accedemos al elemento en el índice 3 usando el método get()
    println(solarSystem.get(3)) // Debería imprimir: Mars
}
```

Earth  
Mars

Kotlin

2.1.20 ▾ JVM ▾ Program arguments

Solutions Docs Community Teach Play

Copy link Share code Run

```
/*
 * Unidad 3
 */
fun main() {
    // Creamos una Lista inmutable con los 8 planetas
    val solarSystem = listOf("Mercury", "Venus", "Earth", "Mars",
        "Jupiter", "Saturn", "Uranus", "Neptune")

    // Buscar el índice de "Earth"
    println(solarSystem.indexOf("Earth")) // Debería imprimir: 2

    // Buscar el índice de "Pluto"
    println(solarSystem.indexOf("Pluto")) // Debería imprimir: -1
}
```

2  
-1

The screenshot shows the Kotlin Play IDE interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the bar, the version is listed as 2.1.20 and the JVM target is selected. A "Program arguments" input field is present. On the right, there are buttons for "Copy link", "Share code", and "Run". The main area contains the following Kotlin code:

```
/*
 * Unidad 3
 */
fun main() {
    // Lista de planetas del sistema solar
    val solarSystem = listOf("Mercury", "Venus", "Earth", "Mars",
        "Jupiter", "Saturn", "Uranus", "Neptune")

    // Recorremos la lista con un bucle for y mostramos cada planeta
    for (planet in solarSystem) {
        println(planet)
    }
}
```

Below the code, the output window displays the names of the planets as they were printed by the program:

```
Mercury
Venus
Earth
Mars
Jupiter
Saturn
Uranus
Neptune
```

En el fragmento de código Kotlin que se muestra abajo, dentro de la función main, se crea una lista mutable llamada solarSystem inicializada con los nombres de los ocho planetas del sistema solar. Posteriormente, se realizan dos operaciones de modificación en esta lista. Primero, se agrega la cadena "Pluto" al final de la lista utilizando el método add(). Luego, se inserta la cadena "Theia" en la posición de índice 3 de la lista, lo que significa que se coloca como el cuarto elemento, desplazando los elementos siguientes. La salida del programa muestra la lista resultante después de estas modificaciones, donde se puede observar que "Pluto" se ha añadido al final y "Theia" se ha insertado en la posición indicada, alterando el orden original de los planetas después de la Tierra y Venus.

The screenshot shows the Kotlin Play IDE interface. The code editor displays the following Kotlin code:

```
/*
 * Unidad 3
 */
fun main() {
    // Lista mutable de planetas
    val solarSystem = mutableListOf(
        "Mercury", "Venus", "Earth", "Mars",
        "Jupiter", "Saturn", "Uranus", "Neptune"
    )

    // Agregamos a Plutón al final de la lista
    solarSystem.add("Pluto")

    // Insertamos a Theia en la posición 3 (antes de Marte)
    solarSystem.add(3, "Theia")
}
```

A tooltip is visible over the line `solarSystem.add(3, "Theia")`, containing the text: "Insertamos a Theia en La posición 3 (antes de Marte)". Below the code editor, a list of planet names is displayed in a dropdown or suggestion list:

- Mercury
- Venus
- Earth
- Theia
- Mars
- Jupiter
- Saturn
- Uranus
- Neptune
- Pluto

En este otro código Kotlin, dentro de la función `main`, se crea una lista mutable llamada `solarSystem` que contiene los nombres de algunos planetas, incluyendo "Theia" y "Pluto". Posteriormente, se realiza una modificación en esta lista: se actualiza el valor del elemento que se encuentra en el índice 3 (la cuarta posición en la lista) a la cadena "Future Moon". Finalmente, se imprimen en la consola los elementos que se encuentran en los índices 3 y 9 de la lista `solarSystem`. La salida del programa muestra "Future Moon", que es el valor asignado al índice 3, y "Pluto", que se encuentra en el índice 9 de la lista. Esto ilustra cómo se puede modificar un elemento específico en una lista mutable utilizando su índice y cómo se pueden acceder a elementos en posiciones particulares para su posterior uso o visualización.

The screenshot shows the Kotlin Play IDE interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the navigation bar, the main area has tabs for '2.1.20' and 'JVM'. A 'Program arguments' input field is present. On the right side, there are buttons for 'Copy link', 'Share code', and a purple 'Run' button.

```
fun main() {
    // Lista mutable con planetas
    val solarSystem = mutableListOf(
        "Mercury", "Venus", "Earth", "Theia", "Mars",
        "Jupiter", "Saturn", "Uranus", "Neptune", "Pluto"
    )

    // Actualizamos el valor en el índice 3
    solarSystem[3] = "Future Moon"

    // Imprimimos los elementos en el índice 3 y 9
    println(solarSystem[3])
    println(solarSystem[9])
}
```

The output window below the code editor shows the results of the execution:

```
Future Moon
Pluto
```

This screenshot shows the same or a very similar setup in the Kotlin Play IDE. The interface and code editor look identical to the first one. The output window at the bottom shows the results of the execution.

```
/*
 * Unidad 3
 */
fun main() {
    // Lista mutable con planetas incluyendo Theia y Pluto
    val solarSystem = mutableListOf(
        "Mercury", "Venus", "Earth", "Future Moon", "Mars",
        "Jupiter", "Saturn", "Uranus", "Neptune", "Pluto"
    )

    // Quitar a "Pluto" por indice
    solarSystem.removeAt(9)

    // Quitar a "Future Moon" por nombre
    solarSystem.remove("Future Moon")
```

The output window shows the results of the execution:

```
false
false
```

The screenshot shows a Kotlin code editor interface. The code creates a mutable set of planets and adds 'Pluto' to it. The output window shows the size of the set before and after adding 'Pluto'.

```
/*
 * Unidad 3
 */
fun main() {
    // Crear un conjunto mutable de planetas
    val solarSystem = mutableSetOf("Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune")

    // Imprimir el tamaño inicial del conjunto
    println(solarSystem.size) // Debería imprimir 8

    // Agregar "Pluto" al conjunto
    solarSystem.add("Pluto")

    // Imprimir el tamaño después de agregar "Pluto"
    println(solarSystem.size) // Debería imprimir 9
}

8
9
true
9
```

The screenshot shows a Kotlin code editor interface. The code creates a mutable map of planet moons and removes 'Pluto' from it. The output window shows the size of the map before and after removing 'Pluto'.

```
/*
 * Unidad 3
 */
fun main() {
    // Crear un conjunto mutable de planetas
    val solarSystem = mutableSetOf("Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune")

    // Agregar "Pluto" al conjunto
    solarSystem.add("Pluto")

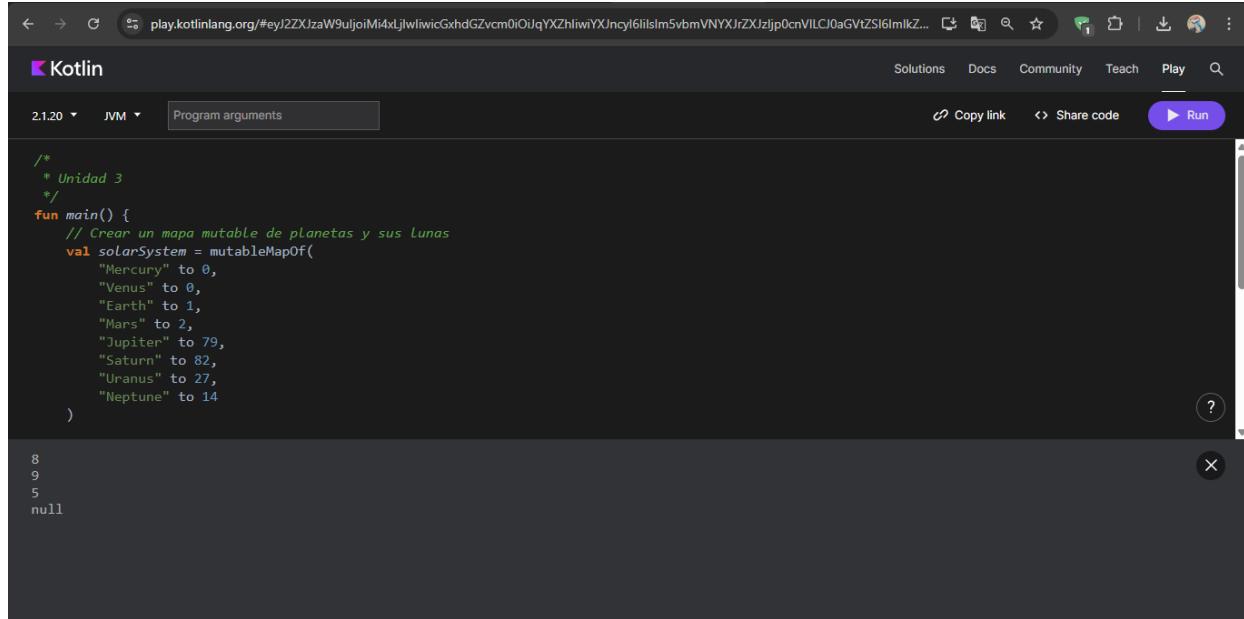
    // Usar remove() para quitar "Pluto" del conjunto
    solarSystem.remove("Pluto")

    // Imprimir el tamaño después de quitar "Pluto"
    println(solarSystem.size) // Debería imprimir 8
}

8
false
```

También podemos observar a continuación que dentro de la función main, se inicializa un mapa mutable llamado solarSystem donde las claves son los nombres de los planetas y los valores son sus respectivos números de lunas. Luego, se realizan varias operaciones de impresión. Primero, se imprime el tamaño del mapa, que es 8. Después, se intenta acceder a las lunas de "Pluto" y "Theia", que no están en el mapa, resultando en la impresión de null para ambos. A continuación, se imprimen las lunas de "Earth" (1) y "Mars" (2). Se vuelve a intentar acceder a las lunas de "Pluto" (imprimiendo null). Finalmente, se imprimen las lunas de "Saturn" (82) y "Uranus" (27). El código demuestra cómo crear un mapa mutable y acceder

a sus valores por clave, mostrando que, si una clave no existe, el acceso devuelve null.

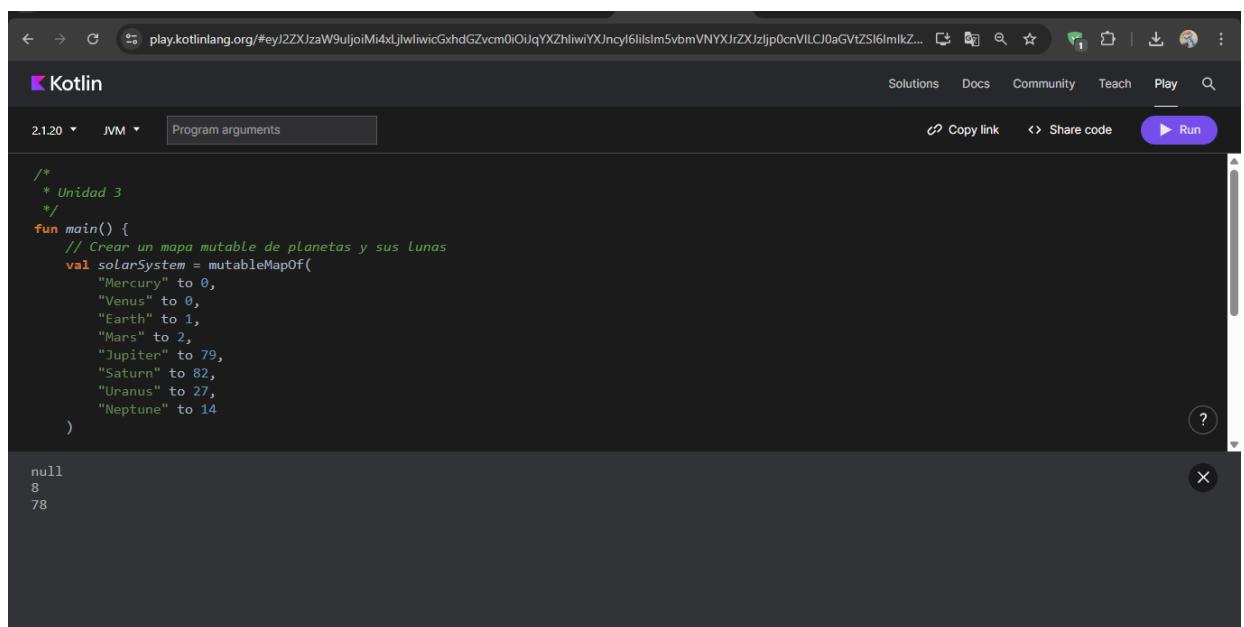


The screenshot shows the Kotlin Play IDE interface. The code in the editor is:

```
/*
 * Unidad 3
 */
fun main() {
    // Crear un mapa mutable de planetas y sus Lunas
    val solarSystem = mutableMapOf(
        "Mercury" to 0,
        "Venus" to 0,
        "Earth" to 1,
        "Mars" to 2,
        "Jupiter" to 79,
        "Saturn" to 82,
        "Uranus" to 27,
        "Neptune" to 14
    )
}
```

The output window below the editor shows the results of running the program:

```
8
9
5
null
```



The screenshot shows the Kotlin Play IDE interface. The code in the editor is identical to the one in the previous screenshot:

```
/*
 * Unidad 3
 */
fun main() {
    // Crear un mapa mutable de planetas y sus Lunas
    val solarSystem = mutableMapOf(
        "Mercury" to 0,
        "Venus" to 0,
        "Earth" to 1,
        "Mars" to 2,
        "Jupiter" to 79,
        "Saturn" to 82,
        "Uranus" to 27,
        "Neptune" to 14
    )
}
```

The output window below the editor shows the results of running the program:

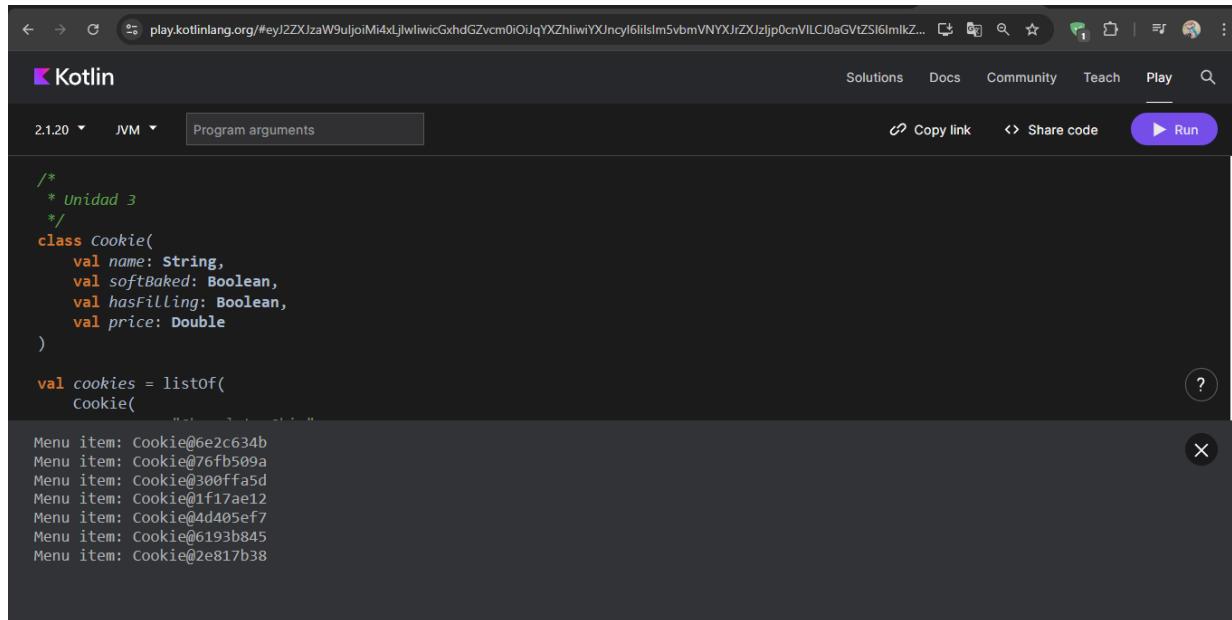
```
null
8
78
```

## Funciones de orden superior con colecciones

En este codelab se define una clase llamada Cookie con propiedades como nombre, si está horneada suave, si tiene relleno y su precio. Luego, se crea una lista inmutable llamada cookies que contiene varias instancias de la clase Cookie, representando diferentes tipos de galletas con sus respectivas características y precios.

Este código Kotlin define una clase llamada Cookie con propiedades como nombre, si está horneada suave, si tiene relleno y su precio. Luego, se crea una lista inmutable llamada cookies que contiene varias instancias de la clase Cookie, representando diferentes tipos de galletas con sus respectivas características y precios. A continuación, se presentan diferentes bloques de código comentados, cada uno ilustrando el uso de una función de colección diferente en Kotlin para manipular esta lista de galletas. Estas funciones incluyen forEach para iterar sobre la lista, map para transformar los elementos, filter para seleccionar elementos basados en una condición, groupBy para agrupar elementos según una característica, fold para acumular un valor a través de la lista, y sortedBy para ordenar la lista según una propiedad.

Se utiliza la función de extensión forEach en la lista cookies. La función forEach itera sobre cada elemento de la lista y ejecuta la lambda proporcionada para cada elemento. En este caso, la lambda simplemente imprime cada objeto Cookie completo utilizando su representación `toString()`. La salida mostrará cada galleta con todos sus atributos.

A screenshot of the Kotlin playground interface. The top navigation bar includes links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the bar, there are dropdown menus for version (2.1.20) and JVM, and a "Program arguments" input field. On the right side, there are buttons for Copy link, Share code, and Run. The main workspace shows the following Kotlin code:

```
/*
 * Unidad 3
 */
class Cookie(
    val name: String,
    val softBaked: Boolean,
    val hasFilling: Boolean,
    val price: Double
)

val cookies = listOf(
    Cookie(
        name = "Galleta de chocolate",
        softBaked = true,
        hasFilling = false,
        price = 1.5
    ),
    Cookie(
        name = "Galleta de vainilla",
        softBaked = false,
        hasFilling = false,
        price = 1.0
    ),
    Cookie(
        name = "Galleta de limón",
        softBaked = true,
        hasFilling = true,
        price = 2.0
    ),
    Cookie(
        name = "Galleta de mantequilla",
        softBaked = false,
        hasFilling = false,
        price = 1.2
    ),
    Cookie(
        name = "Galleta de jengibre",
        softBaked = true,
        hasFilling = false,
        price = 1.8
    )
)
```

The output window at the bottom displays the printed names of the cookies:

```
Menu item: Galleta de chocolate
Menu item: Galleta de vainilla
Menu item: Galleta de limón
Menu item: Galleta de mantequilla
Menu item: Galleta de jengibre
```

En el siguiente, dentro de la lambda, en lugar de imprimir el objeto Cookie completo, se accede específicamente a la propiedad `name` de cada galleta y se imprime. La salida será una lista de solo los nombres de cada galleta.

The screenshot shows the Kotlin playground interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the bar, the version is set to 2.1.20 and the JVM target is selected. A "Program arguments" input field contains the string "cookies". The main code area contains a class definition for a cookie and a main function that prints each cookie's name. The output window at the bottom shows the names of seven different cookie instances.

```
2.1.20 ▾ JVM ▾ Program arguments

class Cookie {
    val name = "Sugar and Sprinkles",
    softBaked = false,
    hasFilling = false,
    price = 1.39
}

fun main() {
    cookies.forEach {
        println("Menu item: ${it.name}")
    }
}

Menu item: Cookie@6e2c634b.name
Menu item: Cookie@76fb509a.name
Menu item: Cookie@300ffa5d.name
Menu item: Cookie@1f17ae12.name
Menu item: Cookie@4d405ef7.name
Menu item: Cookie@6193b845.name
Menu item: Cookie@2e817b38.name
```

This screenshot is similar to the one above, but the code has been modified. It includes a class definition for a cookie with properties name, softBaked, hasFilling, and price. A variable cookies is defined as a list of these objects. The main function prints each cookie's name. The output window shows the names of seven different cookie instances.

```
2.1.20 ▾ JVM ▾ Program arguments

class Cookie(
    val name: String,
    val softBaked: Boolean,
    val hasFilling: Boolean,
    val price: Double
)

val cookies = listOf(
    Cookie(
        name = "Chocolate Chip",
        softBaked = true,
        hasFilling = false,
        price = 1.5
    ),
    Cookie(
        name = "Banana Walnut",
        softBaked = true,
        hasFilling = false,
        price = 1.5
    ),
    Cookie(
        name = "Vanilla Creme",
        softBaked = true,
        hasFilling = true,
        price = 1.5
    ),
    Cookie(
        name = "Chocolate Peanut Butter",
        softBaked = true,
        hasFilling = true,
        price = 1.5
    ),
    Cookie(
        name = "Snickerdoodle",
        softBaked = true,
        hasFilling = false,
        price = 1.5
    ),
    Cookie(
        name = "Blueberry Tart",
        softBaked = false,
        hasFilling = true,
        price = 1.5
    ),
    Cookie(
        name = "Sugar and Sprinkles",
        softBaked = false,
        hasFilling = false,
        price = 1.5
    )
)

cookies.forEach {
    println("Menu item: ${it.name}")
}
```

También se introduce la función de extensión map, la cual transforma cada elemento de la lista original en un nuevo elemento según la lambda proporcionada y devuelve una nueva lista con estos elementos transformados. En este caso, la lambda toma cada objeto Cookie y crea una cadena que contiene el nombre de la galleta y su precio, formateado como "nombre - \$precio". La lista resultante fullMenu se imprime, mostrando el menú completo con el nombre y el precio de cada galleta.

The screenshot shows the Kotlin Play IDE interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, and Play. Below the navigation bar, there are dropdown menus for '2.1.20' and 'JVM'. A 'Program arguments' input field is present. On the right side, there are buttons for 'Copy link', 'Share code', and 'Run'. The main area contains Java code:

```
/*
 * Unidad 3
 */

class Cookie(
    val name: String,
    val softBaked: Boolean,
    val hasFilling: Boolean,
    val price: Double
)

val cookies = listOf(
    ...
)

Full menu:
Chocolate Chip - $1.69
Banana Walnut - $1.49
Vanilla Creme - $1.59
Chocolate Peanut Butter - $1.49
Snickerdoodle - $1.39
Blueberry Tart - $1.79
Sugar and Sprinkles - $1.39
```

A tooltip is displayed over the word 'cookies', listing the items from the 'Full menu':

- Chocolate Chip - \$1.69
- Banana Walnut - \$1.49
- Vanilla Creme - \$1.59
- Chocolate Peanut Butter - \$1.49
- Snickerdoodle - \$1.39
- Blueberry Tart - \$1.79
- Sugar and Sprinkles - \$1.39

Por otra parte, la función de extensión filter se utiliza para crear una nueva lista que contiene solo los elementos de la lista original que cumplen con la condición especificada en la lambda. En el caso de uno de los códigos de abajo la lambda `it.softBaked` verifica si la propiedad `softBaked` de cada `Cookie` es true. La lista resultante `softBakedMenu` contendrá solo las galletas que están horneadas suaves, y luego se imprime el nombre y el precio de cada una de estas galletas. Mientras que el que usa la función de extensión `groupBy` agrupa los elementos de la lista original según el valor devuelto por la lambda proporcionada y devuelve un mapa donde las claves son los valores de agrupación y los valores son listas de los elementos que pertenecen a cada grupo. Aquí, la lambda `it.softBaked` agrupa las galletas en dos grupos: true (horneadas suaves) y false (no horneadas suaves). Luego, se acceden a estos grupos desde el mapa `groupedMenu` y se imprimen los nombres y precios de las galletas en cada grupo (`softBakedMenu` y `crunchyMenu`).

```
/*
 * Unidad 3
 */

class Cookie(
    val name: String,
    val softBaked: Boolean,
    val hasFilling: Boolean,
    val price: Double
)

val cookies = listOf(
    ...
)
```

Soft cookies:  
Banana Walnut - \$1.49  
Snickerdoodle - \$1.39  
Blueberry Tart - \$1.79

```
)
```

```
fun main() {
    val groupedMenu = cookies.groupBy { it.softBaked }

    val softBakedMenu = groupedMenu[true] ?: emptyList()
    val crunchyMenu = groupedMenu[false] ?: emptyList()

    println("Soft cookies:")
    softBakedMenu.forEach {
        println("${it.name} - ${it.price}")
    }
    println("Crunchy cookies:")
}
```

Soft cookies:  
Banana Walnut - \$1.49  
Snickerdoodle - \$1.39  
Blueberry Tart - \$1.79  
Crunchy cookies:  
Chocolate Chip - \$1.69  
Vanilla Creme - \$1.59  
Chocolate Peanut Butter - \$1.49  
Sugar and Sprinkles - \$1.39

La función de extensión `fold` se utiliza para acumular un valor a través de todos los elementos de la lista. Toma un valor inicial (aquí 0.0) y una lambda que se ejecuta para cada elemento de la lista, tomando el valor acumulado actual y el elemento actual como argumentos y devolviendo el nuevo valor acumulado. En este caso, la lambda suma el precio de cada `Cookie` al total acumulado. El resultado final `totalPrice` es la suma de los precios de todas las galletas en la lista, y se imprime.

A screenshot of the Kotlin playground interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the navigation bar, the version is set to 2.1.20 and the JVM target is selected. A "Program arguments" input field is present. On the right side, there are buttons for "Copy link", "Share code", and "Run". The main area contains the following Kotlin code:

```
class Cookie(
    val name: String,
    val softBaked: Boolean,
    val hasFilling: Boolean,
    val price: Double
)

val cookies = listOf(
    Cookie(
        name = "Chocolate Chip",
        softBaked = false,
        hasFilling = false,
        ...
    )
)

```

Below the code, the output is displayed:

```
Total price: $10.83
```

Finalmente, la función de extensión `sortedBy` se utiliza para ordenar los elementos de la lista según el valor devuelto por la lambda proporcionada. Devuelve una nueva lista con los elementos ordenados. Aquí, la lambda `it.name` indica que la lista de galletas debe ordenarse alfabéticamente por su nombre. La lista resultante `alphabeticalMenu` se imprime, mostrando solo el nombre de cada galleta en orden alfabético.

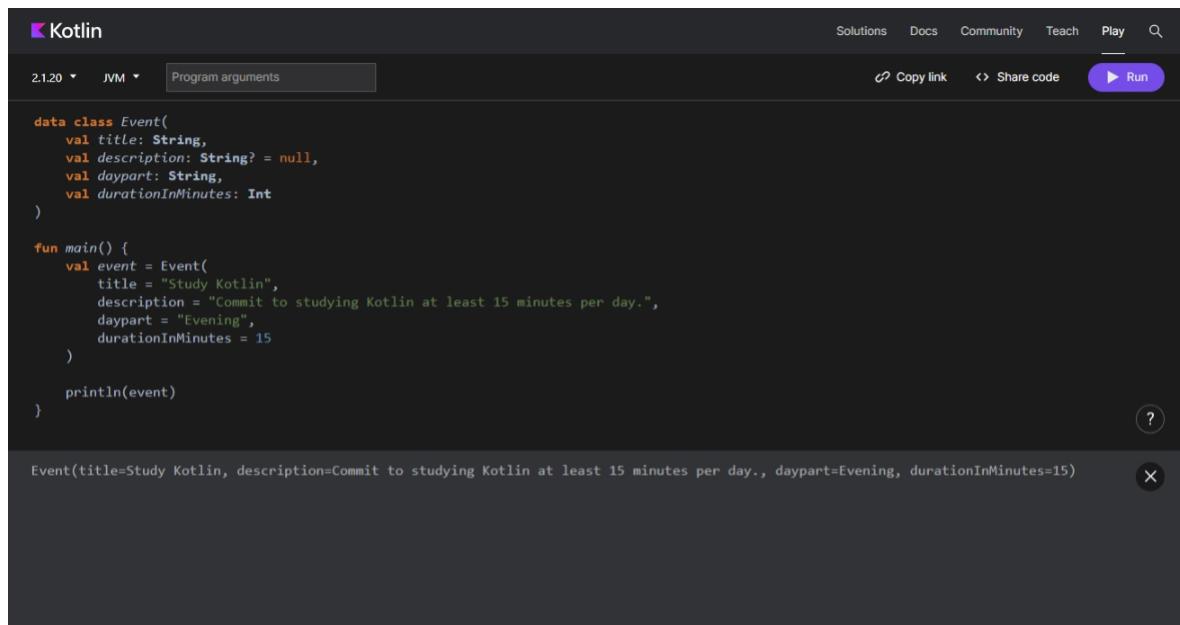
A screenshot of the Kotlin playground interface, identical to the previous one in layout and version. The code is the same as above, but the output is different:

```
Alphabetical menu:
Banana Walnut
Blueberry Tart
Chocolate Chip
Chocolate Peanut Butter
Snickerdoodle
Sugar and Sprinkles
Vanilla Creme
```

## Práctica: Clases y colecciones

Aquí se trabaja con clases, propiedades, constructores, y colecciones como listas y mapas, resolviendo una serie de tareas donde cada una plantea un pequeño reto, como crear clases con atributos, recorrer listas, filtrar datos o transformar colecciones, y luego se ofrece una solución guiada.

A lo largo de las tareas aprendemos a crear clases de datos como Event, refactorizar atributos utilizando enumeraciones (enum class), y trabajar con listas mutables. También se exploran funciones comunes como filter para seleccionar eventos cortos, groupBy para agruparlos según la parte del día, y métodos como last() para acceder al último evento.

A screenshot of the Kotlin Play IDE interface. The top navigation bar includes 'Solutions', 'Docs', 'Community', 'Teach', 'Play' (which is selected), and a search icon. Below the bar, there are dropdown menus for '2.1.20' and 'JVM', and a 'Program arguments' input field. On the right, there are 'Copy link', 'Share code', and a 'Run' button. The main code editor area contains the following Kotlin code:

```
data class Event(
    val title: String,
    val description: String? = null,
    val daypart: String,
    val durationInMinutes: Int
)

fun main() {
    val event = Event(
        title = "Study Kotlin",
        description = "Commit to studying Kotlin at least 15 minutes per day.",
        daypart = "Evening",
        durationInMinutes = 15
    )
    println(event)
}
```

The output window below the editor shows the result of running the code:

```
Event(title=Study Kotlin, description=Commit to studying Kotlin at least 15 minutes per day., daypart=Evening, durationInMinutes=15)
```

Tarea 1. Crear una clase de datos (data class) en Kotlin para almacenar información

The screenshot shows the Kotlin Play IDE interface. The code in the editor is:

```
enum class Daypart {
    MORNING,
    AFTERNOON,
    EVENING
}

data class Event(
    val title: String,
    val description: String? = null,
    val daypart: Daypart,
    val durationInMinutes: Int
)

fun main() {
    val event = Event(
        title = "Study Kotlin",
        description = "Commit to studying Kotlin at least 15 minutes per day.",
        daypart = Daypart.EVENING,
        durationInMinutes = 15
    )

    println(event)
}
```

A tooltip is displayed below the code, showing the output of the println statement:

Event(title=Study Kotlin, description=Commit to studying Kotlin at least 15 minutes per day., daypart=EVENING, durationInMinutes=15)

Tarea 2. Cómo usar enum class para evitar errores con strings inconsistentes.

The screenshot shows the Kotlin Play IDE interface. The code in the editor is:

```
// Enum para representar las franjas del día
enum class Daypart {
    MORNING,
    AFTERNOON,
    EVENING
}

// Clase de datos para eventos
data class Event(
    val title: String,
    val description: String? = null,
    val daypart: Daypart,
    val durationInMinutes: Int,
)

fun main() {
    // Crear eventos individuales
    val event1 = Event(title = "Wake up", description = "Time to get up", daypart = Daypart.MORNING, durationInMinutes = 0)
    val event2 = Event(title = "Eat breakfast", daypart = Daypart.MORNING, durationInMinutes = 15)
    val event3 = Event(title = "Learn about Kotlin", daypart = Daypart.AFTERNOON, durationInMinutes = 30)
    val event4 = Event(title = "Practice Compose", daypart = Daypart.AFTERNOON, durationInMinutes = 60)
    val event5 = Event(title = "Watch latest Devbytes video", daypart = Daypart.AFTERNOON, durationInMinutes = 10)
    val event6 = Event(title = "Check out latest Android Jetpack library", daypart = Daypart.EVENING, durationInMinutes = 45)

    // Lista mutable de eventos
    val eventos = mutableListOf(event1, event2, event3, event4, event5, event6)
```

Tarea 3. Organizar eficientemente los datos usando una lista mutable (mutableListOf) para almacenar todos los eventos

Kotlin

Solutions Docs Community Teach Play

2.1.20 ▾ JVM ▾ Program arguments

Copy link Share code Run

```
val event5 = Event(title = "Watch latest DevBytes video", daypart = Daypart.AFTERNOON, durationInMinutes = 10)
val event6 = Event(title = "Check out latest Android Jetpack library", daypart = Daypart.EVENING, durationInMinutes = 45)

// Lista mutable de eventos
val events = mutableListOf(event1, event2, event3, event4, event5, event6)

// Mostrar cantidad de eventos
println("Total events scheduled: ${events.size}")

// Mostrar todos los eventos
events.forEach {
    println(it)
}
```

Total events scheduled: 6

Event(title=Wake up, description=Time to get up, daypart=MORNING, durationInMinutes=0)  
Event(title=Eat breakfast, description=null, daypart=MORNING, durationInMinutes=15)  
Event(title=Learn about Kotlin, description=null, daypart=AFTERNOON, durationInMinutes=30)  
Event(title=Practice Compose, description=null, daypart=AFTERNOON, durationInMinutes=60)  
Event(title=Watch latest DevBytes video, description=null, daypart=AFTERNOON, durationInMinutes=10)  
Event(title=Check out latest Android Jetpack library, description=null, daypart=EVENING, durationInMinutes=45)

### Tarea 3. 1 Ejecución

Kotlin

Solutions Docs Community Teach Play

2.1.20 ▾ JVM ▾ Program arguments

Copy link Share code Run

```
// Mostrar cantidad total de eventos
println("Total events scheduled: ${events.size}")

// Mostrar todos los eventos
events.forEach {
    println(it)
}

println() // Línea en blanco para separar

// Filtrar eventos cortos (menos de 60 min) y mostrar cantidad
val shortEvents = events.filter { it.durationInMinutes < 60 }
println("You have ${shortEvents.size} short events.")
```

Total events scheduled: 6

Event(title=Wake up, description=Time to get up, daypart=MORNING, durationInMinutes=0)  
Event(title=Eat breakfast, description=null, daypart=MORNING, durationInMinutes=15)  
Event(title=Learn about Kotlin, description=null, daypart=AFTERNOON, durationInMinutes=30)  
Event(title=Practice Compose, description=null, daypart=AFTERNOON, durationInMinutes=60)  
Event(title=Watch latest DevBytes video, description=null, daypart=AFTERNOON, durationInMinutes=10)  
Event(title=Check out latest Android Jetpack library, description=null, daypart=EVENING, durationInMinutes=45)

You have 5 short events.

### Tarea 4. Se agrega un filtro para los eventos cortos (menos de 60 min) y se muestra la cantidad

The screenshot shows the Kotlin Play IDE interface. At the top, there are dropdown menus for '2.1.20' and 'JVM'. A 'Program arguments' input field contains no text. On the right, there are buttons for 'Copy link', 'Share code', and a purple 'Run' button. Below the interface is a code editor window containing the following Kotlin code:

```
// Mostrar resumen de eventos cortos
val shortEvents = events.filter { it.durationInMinutes < 60 }
println("You have ${shortEvents.size} short events.\n")

// Agrupar eventos por franja horaria y mostrar resumen
val groupedEvents = events.groupBy { it.daypart }
groupedEvents.forEach { (daypart, eventsByDaypart) ->
    println("${daypart.name.lowercase().replaceFirstChar { it.uppercase() }}: ${eventsByDaypart.size} events")
}

You have 5 short events.

Morning: 2 events
Afternoon: 3 events
Evening: 1 events
```

*Tarea 5. Nueva funcionalidad añadida al final, usando el groupBy para agrupar los eventos por franja horaria y mostrar el resumen.*

The screenshot shows the same Kotlin Play IDE interface as the previous one. The code has been modified to include a call to the `last()` function to print the last event of the day. The code now looks like this:

```
// Mostrar resumen de eventos cortos
val shortEvents = events.filter { it.durationInMinutes < 60 }
println("You have ${shortEvents.size} short events.\n")

// Agrupar eventos por franja horaria y mostrar resumen
val groupedEvents = events.groupBy { it.daypart }
groupedEvents.forEach { (daypart, eventsByDaypart) ->
    println("${daypart.name.lowercase().replaceFirstChar { it.uppercase() }}: ${eventsByDaypart.size} events")
}

println("\nLast event of the day: ${events.last().title}")
}

You have 5 short events.

Morning: 2 events
Afternoon: 3 events
Evening: 1 events

Last event of the day: Check out latest Android Jetpack library
```

*Tarea 6. Implementación de la función last() para obtener el último elemento de una lista*

The screenshot shows the Kotlin IDE interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the navigation bar, there are dropdown menus for version (2.1.20) and JVM, and a "Program arguments" input field. To the right of these are buttons for "Copy link", "Share code", and a purple "Run" button. The main area contains code in a dark-themed editor. The code defines a class with a companion object containing a function that prints the duration of the first event of the day. It also includes an extension property `durationOfEvent` for the `Event` class that returns either "short" or "long" based on the event's duration in minutes. Below the code, the terminal window shows the execution results: "You have 5 short events.", followed by counts for Morning (2), Afternoon (3), and Evening (1) events. The last event of the day is noted as "Check out latest Android Jetpack library". The duration of the first event is identified as "short".

```
// Llama a la propiedad de extensión que se creó para obtener si ese evento es "short" o "Long"
// (según su duración).
println("\nDuration of first event of the day: ${events[0].durationOfEvent}")

}

// Se crea una propiedad de extensión llamada durationOfEvent para la clase Event para darle una descripción
// (en este caso "short" o "Long")
val Event.durationOfEvent: String
    get() = if (this.durationInMinutes < 60) {
        "short"
    } else {
        "long"
    }

You have 5 short events.

Morning: 2 events
Afternoon: 3 events
Evening: 1 events

Last event of the day: Check out latest Android Jetpack library

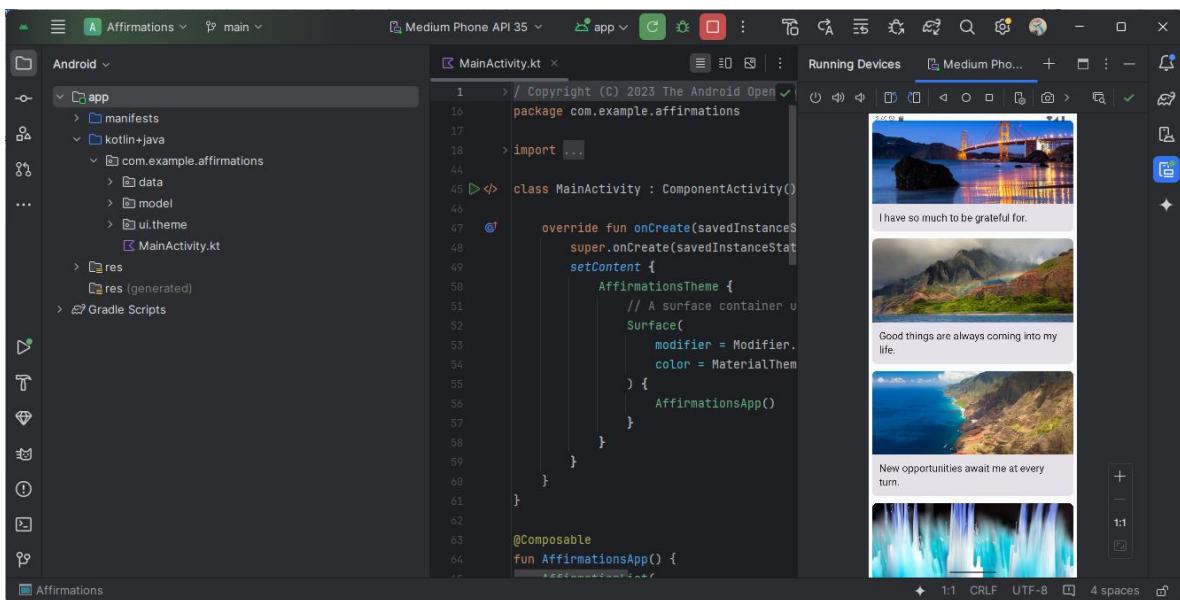
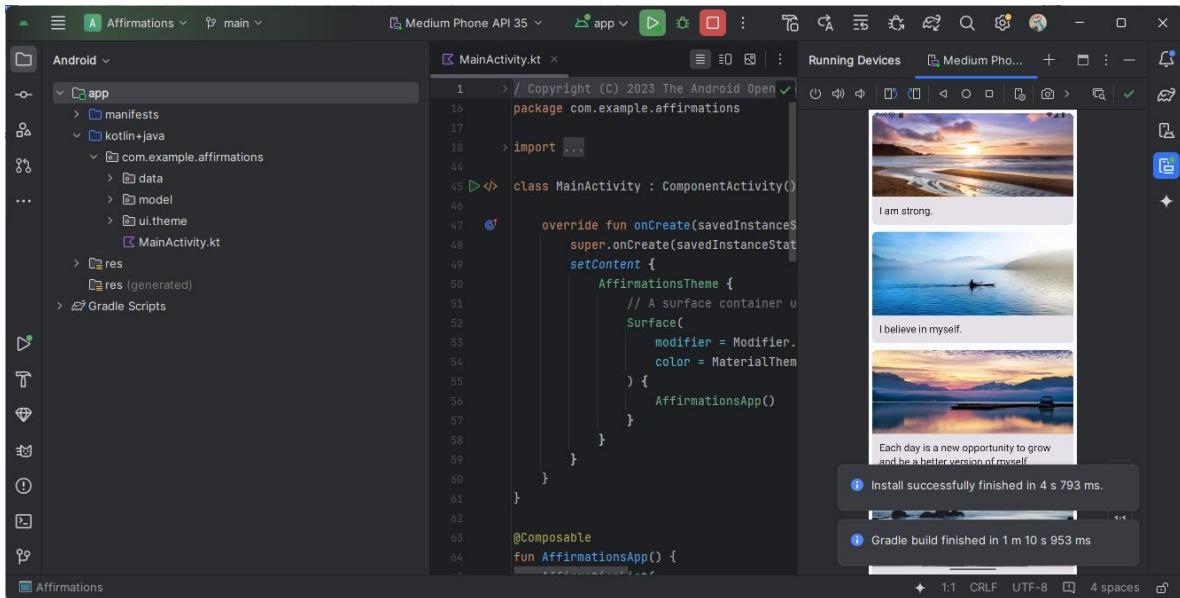
Duration of first event of the day: short
```

Tarea 7. Se crea una propiedad de extensión llamada `durationOfEvent` para la clase `Event` para obtener si ese evento es "short" o "long"

### Ruta de Aprendizaje 2 (Crea una lista desplazable)

#### ¿Cómo agregar una lista desplazable?

En este codelab se utiliza Jetpack Compose para mostrar una lista desplazable de afirmaciones, donde cada afirmación consiste en una imagen y un texto. La clase de datos `Affirmation` define la estructura de cada elemento de la lista, conteniendo IDs de recursos para la cadena y la imagen. La clase `Datasource` se encarga de crear y proporcionar la lista de objetos `Affirmation`, cargando las referencias a los recursos de la aplicación. En la interfaz de usuario (`MainActivity.kt`), el composable `AffirmationCard` define cómo se muestra cada elemento individual dentro de una tarjeta, incluyendo una imagen y un texto con estilos definidos. El composable `AffirmationList` utiliza `LazyColumn` para mostrar de manera eficiente la lista de afirmaciones, solo componiendo los elementos visibles en pantalla y permitiendo el desplazamiento. Finalmente, el composable `AffirmationsApp` ensambla toda la interfaz, obteniendo los datos del `Datasource` y mostrando la lista dentro de una `Surface` con el tema de la aplicación aplicado. También se incluye un composable `AffirmationCardPreview` para facilitar la visualización del diseño de un solo elemento en el entorno de desarrollo.

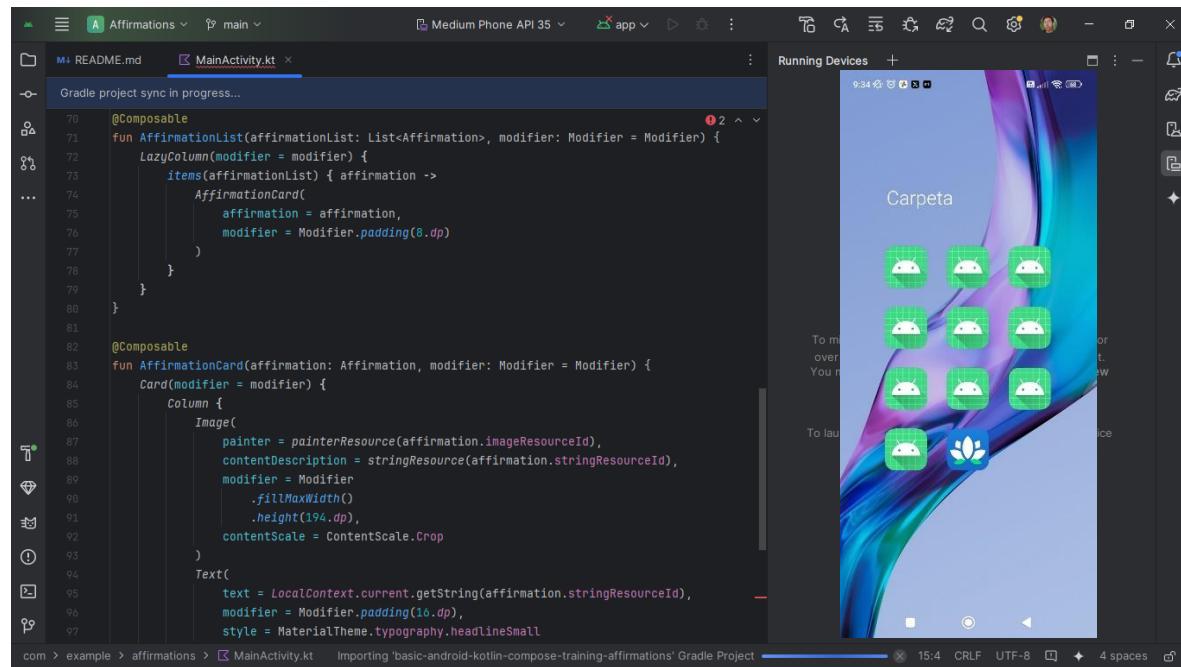
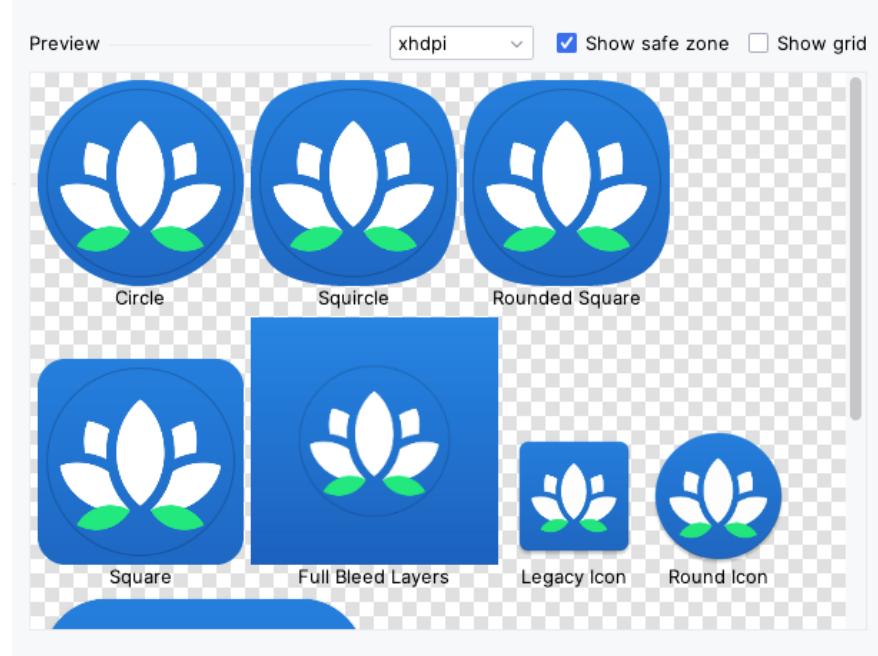


## Cómo cambiar el ícono de la app

Para personalizar el ícono de la aplicación y reemplazar el logo predeterminado de Android, se llevó a cabo el proceso de configuración utilizando los recursos gráficos adecuados. Primero, se colocaron los archivos de íconos en las carpetas `mipmap` correspondientes, generando distintas versiones del ícono (`mdpi`, `hdpi`, `xhdpi`, `xxhdpi` y `xxxhdpi`) para asegurar compatibilidad con distintos tamaños y densidades de pantalla. También se incluyeron calificadores de recursos (`v24`, `v26`, etc.) para aplicar versiones específicas del ícono en dispositivos con diferentes niveles de API.

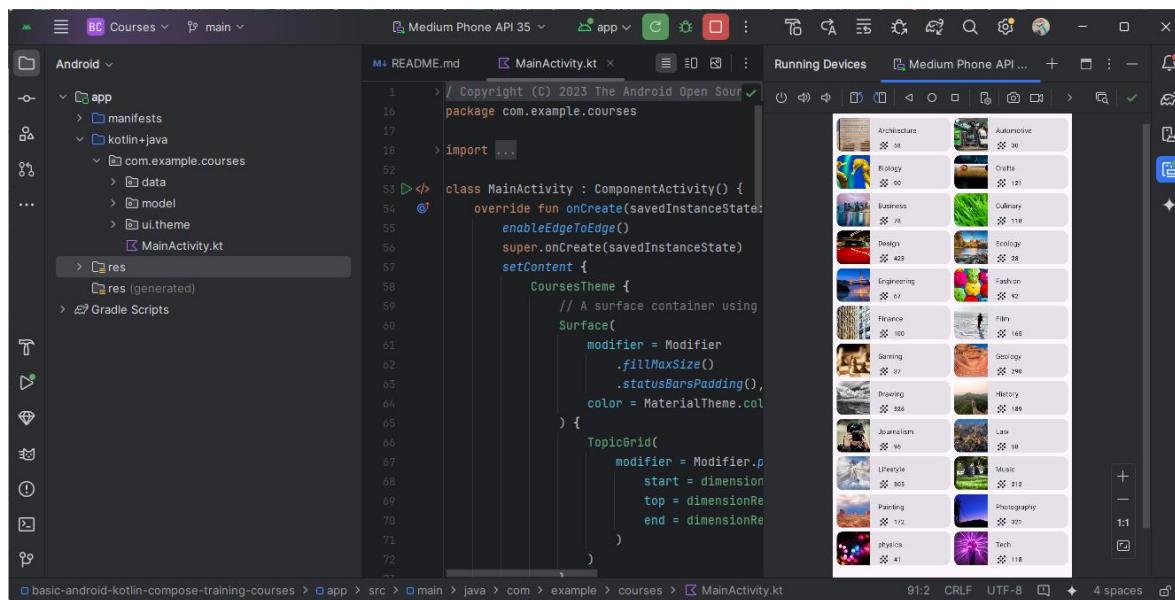
Además, se utilizaron elementos gráficos vectoriales definidos en XML, lo que permite escalabilidad sin pérdida de calidad. Implementé íconos adaptables, introducidos desde el nivel de API 26, que están formados por una capa de fondo y una de primer plano, lo que mejora la apariencia del ícono en distintos dispositivos y launchers.

Finalmente, se manejó el uso de Image Asset Studio, una herramienta integrada en Android Studio, para generar de forma sencilla los íconos heredados y adaptables, asegurando así que la aplicación tenga un ícono personalizado de alta calidad.



## Práctica: Crea una cuadrícula

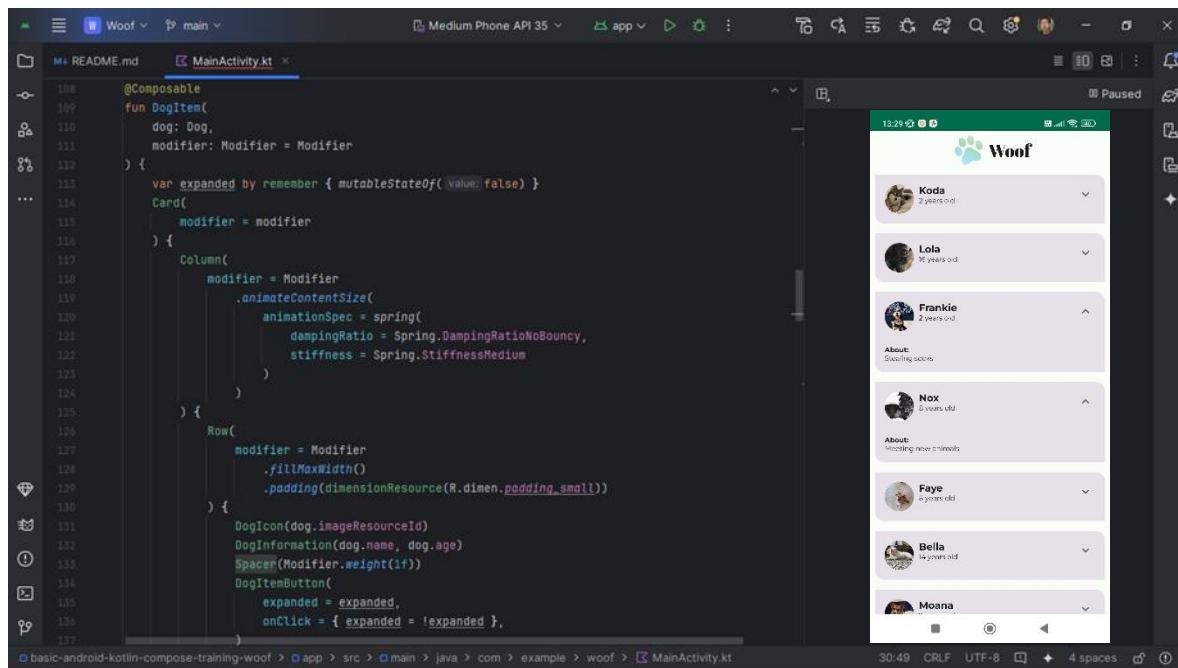
La app tiene como objetivo mostrar una lista de temas de cursos organizada en una cuadrícula. Se crea una clase de datos, una función de componibilidad para representar cada elemento de la cuadrícula y otra función para mostrar la cuadrícula completa. Para ello, contamos con recursos como imágenes por tema y un ícono decorativo, y se utiliza un conjunto de datos que incluye temas como arquitectura, manualidades, negocios, entre otros, junto con su número de cursos e imagen asociada.



## Ruta de Aprendizaje 3 (Compila apps fabulosas)

### Temas de Material con Jetpack Compose - Animación simple con Jetpack Compose

En este codelab se ve acerca de la app Woof, donde se muestra una lista de perros usando Material Design para crear una experiencia de usuario atractiva. El codelab se enfoca en el uso de Temas de Material para mejorar el aspecto de las apps. El archivo `Theme.kt` contiene la información del tema de la app, definido por color, tipografía y forma. Dentro de este archivo, el elemento `WoofTheme()` establece los colores, tipografía y formas de la app. Además, la app utiliza paletas de colores para temas claros y oscuros, y define estilos de tipo específicos. El archivo `Color.kt` incluye los colores usados, que luego se asignan en `Theme.kt` a ranuras específicas para los temas claros y oscuros, mientras que `Shape.kt` define las formas de la app (pequeña, mediana y grande) con opciones para redondear las esquinas. Además, `Type.kt` inicializa las fuentes y asigna `fontFamily`, `fontWeight` y `fontSize` para la escala de tipos de Material Design.



## Práctica: Compila apps de superhéroes

La app de Superhéroes desarrollada en este codelab se centra en la creación de una interfaz de usuario atractiva utilizando los Temas de Material Design y la visualización de una lista de superhéroes. Se realiza la personalización de la apariencia de la app a través de la definición de una paleta de colores para los temas claro y oscuro, la configuración de la tipografía con una fuente personalizada (Cabin) y la definición de formas para los elementos de la interfaz.

En cuanto a la estructura de la app, se introduce la creación de una barra superior utilizando el componente Scaffold, donde se puede agregar un título con un estilo DisplayLarge y alineararlo. También se muestra cómo personalizar el color de la barra de estado para lograr una apariencia de pantalla completa, utilizando la función setUpEdgeToEdge dentro del archivo Theme.kt.

La visualización de la lista de superhéroes se aborda con la creación de un elemento de lista individual en el archivo HeroesScreen.kt, definiendo su diseño con elementos como una imagen (de tamaño fijo y con esquinas redondeadas), el nombre del superhéroe con estilo DisplaySmall y su descripción con estilo BodyLarge, utilizando Box para el diseño y aplicando padding y recorte. Finalmente, se explica cómo crear una lista desplazable eficiente (LazyColumn) que toma la lista de héroes y muestra cada elemento con el diseño previamente definido, aplicando también padding a la lista.

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

- Project Structure:** Shows the project name "Superheroes" and the file "MainActivity.kt".
- Code Editor:** Displays the Kotlin code for MainActivity. The code includes imports for `ComponentActivity`, `Modifier`, `Scaffold`, and `HeroesRepository`. It defines a `TopAppBar` and a `HeroesList` component.
- Rendering Preview:** A window titled "Superheroes" shows a list of heroes with cards. Each card contains a hero's name, a brief description, and a small icon. The heroes listed are:
  - Nick the Night and Day
  - Reality Protector
  - Andre the Giant
  - Benjamin the Brave
  - Magnificent Maru
  - Dynamic Yasmine
- Bottom Bar:** Shows the file path "lin-compose-training-superheroes > app > src > main > java > com > example > superl", the line number "93:15", and encoding information "CRLF UTF-8".

## Proyecto: Crea una app de 30 días

Este proyecto consiste en desarrollar una aplicación Android que ofrezca 30 sugerencias, una para cada día del mes, basadas en un tema elegido por el usuario (en nuestro caso, nos decidimos por una lista de 30 días en donde se mostrarrán ideas motivacionales). Cada una de las 30 sugerencia incluye texto relevante y una imagen, y presenta en una lista desplazable o cuadrícula. En esta aplicación se siguieron los lineamientos de Material Design y cuenta con un diseño único y personalizado. Como ejemplo, la app "30 Days of Wellness" utiliza tarjetas con un indicador de día, un resumen de la sugerencia, una imagen y texto adicional. El objetivo de este proyecto fue aplicar los conceptos aprendidos en la Unidad 3 del curso para crear una aplicación funcional y estilizada, demostrando las habilidades adquiridas en el uso de Jetpack Compose y Material Design.

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

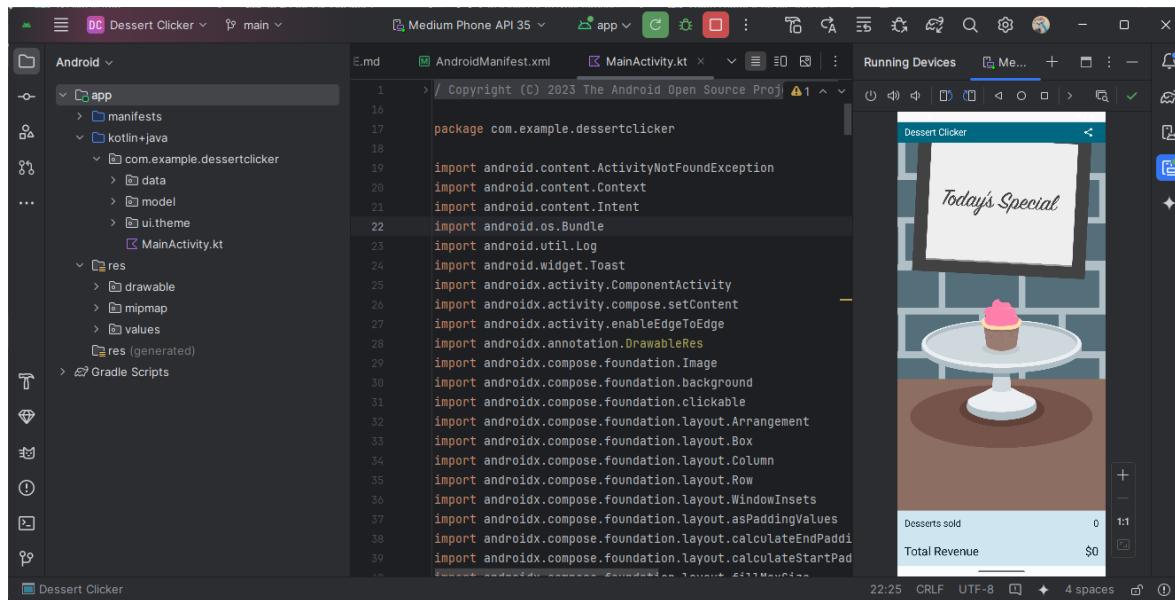
- Project Structure:** Shows files like README.md and MainActivity.kt.
- MainActivity.kt Code:** The code implements a `TopAppBar` with a scroll behavior and a `Scaffold` containing a `LazyColumn`. The `LazyColumn` has vertical arrangement, horizontal alignment, and padding values defined.
- Running Devices:** Displays two preview screens for the app's user interface.
  - Day 29: Be Positive:** Shows a white mug with "BE POSITIVE" text on a pink background. The text below reads: "Everyone has that negative voice inside that makes things feel impossible. Let's shut it up for a moment and remind ourselves that we can - and will - overcome."
  - Day 30: Be Happy:** Shows a pink rock with "SMILE" written on it on a green surface. The text below reads: "Things can always be worse. Just be happy that you may never actually know how much worse."
- Bottom Bar:** Includes icons for zoom, orientation, and other developer tools.

## Unidad 4: Navegación y arquitectura de la app

### Ruta de Aprendizaje 1 (Componentes de la arquitectura)

#### Etapas del ciclo de vida de la actividad

En este codelab se explora a fondo el ciclo de vida de las actividades en Android, ilustrando cómo la aplicación pasa por diferentes estados desde su creación hasta su destrucción. Se demuestra, mediante el uso de Logcat, el orden en que se invocan los métodos del ciclo de vida en diversas situaciones, como el inicio, cierre, paso a segundo plano y la interacción con otras actividades. Un punto crucial que se aborda es cómo los cambios de configuración del dispositivo, especialmente la rotación de pantalla, desencadenan la destrucción y recreación de la actividad, lo que puede llevar a la pérdida del estado de la interfaz de usuario. Para solucionar este problema en el contexto de Jetpack Compose, se introduce la diferencia entre remember y rememberSaveable. Mientras que remember conserva el estado solo durante las recomposiciones dentro del ciclo de vida de un componente, rememberSaveable extiende esta persistencia a través de los cambios de configuración.

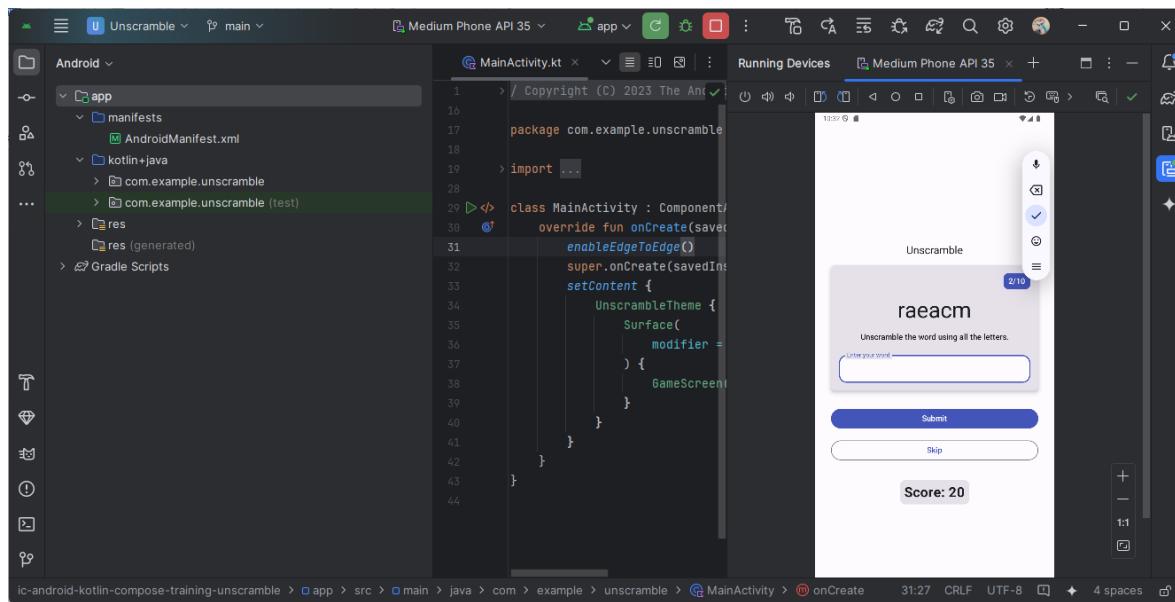


#### ViewModel y el estado en Compose

Este codelab nos guía en el funcionamiento del código del juego Unscramble donde inicialmente la actividad principal (MainActivity) se encarga de cargar y mostrar la interfaz de usuario del juego, la cual está definida en el archivo GameScreen.kt utilizando Jetpack Compose. Este archivo contiene la estructura visual del juego, incluyendo la presentación de la palabra desordenada, el campo de entrada para la respuesta del jugador, los botones de acción ("Submit" y "Skip") y la visualización de la puntuación actual. La lógica central del juego reside en el GameViewModel. Este componente es el encargado de seleccionar las palabras del juego desde

`WordsData.kt`, desordenarlas para presentárselas al jugador, y mantener el estado actual del juego, como la palabra que se está intentando adivinar, la puntuación acumulada y el número de palabras jugadas.

Cuando el jugador interactúa con la interfaz de usuario, por ejemplo, ingresando una respuesta o presionando un botón, estas acciones se comunican al `GameViewModel`. Este último procesa la entrada del jugador, verifica si la respuesta es correcta, actualiza el estado del juego en consecuencia (incrementando la puntuación si la respuesta es correcta, pasando a la siguiente palabra), y notifica a la `GameScreen` sobre cualquier cambio en el estado. Jetpack Compose, al ser un framework de interfaz de usuario reactivo, se encarga de actualizar automáticamente la pantalla para reflejar el nuevo estado del juego. Por ejemplo, si el jugador adivina correctamente una palabra, el `GameViewModel` actualiza la puntuación en su estado, y este cambio se propaga a la `GameScreen`, que recomponerá la sección de la puntuación para mostrar el valor actualizado. De manera similar, el `GameViewModel` determina cuándo se alcanza el final de la partida (basándose en la constante `MAX_NO_OF_WORDS` de `WordsData.kt`) y activa la presentación del diálogo de puntuación final (`FinalScoreDialog`) en la `GameScreen`. En resumen, el `GameViewModel` dirige la lógica del juego, mientras que `GameScreen` se encarga de la presentación visual y la interacción con el usuario, manteniendo ambos componentes comunicados a través del estado del juego.



## Cómo escribir pruebas de unidades para ViewModel

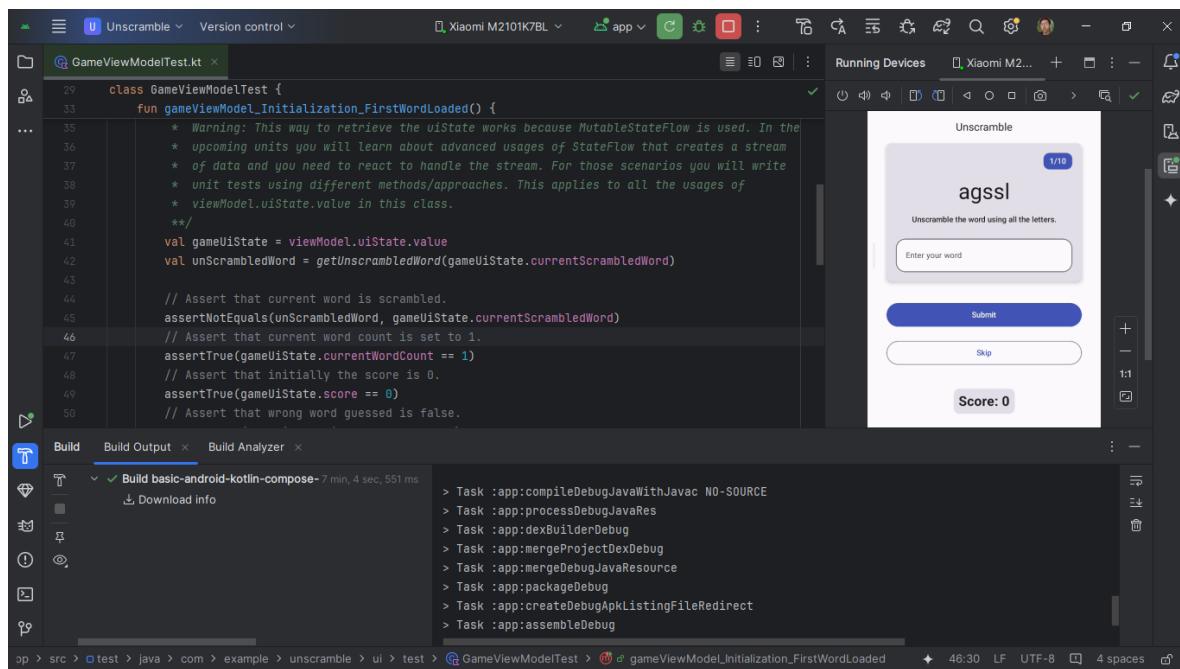
Para garantizar la calidad y confiabilidad de la lógica del juego en la aplicación *Unscramble*, se implementaron pruebas unitarias sobre la clase `GameViewModel`. Estas pruebas siguen una estrategia centrada en cubrir distintas rutas de ejecución del código, abordando los siguientes escenarios clave:

- **Ruta de éxito:** Validamos que el flujo principal del juego se comporta como se espera cuando el usuario proporciona una palabra correcta. Esto incluye la actualización correcta de la puntuación, la cantidad de palabras mostradas y el reinicio del indicador de error.
- **Ruta de error:** Se verifica que, al ingresar una palabra incorrecta, la app no modifique la puntuación y establezca correctamente una bandera de error (isGuessedWordWrong).
- **Caso límite:** Se comprueba que el estado inicial del juego esté configurado correctamente al iniciar una nueva partida. Esto incluye asegurarse de que el contador de palabras comience en 1, la puntuación sea 0, y que no haya errores ni se haya alcanzado el fin del juego.

Además, se aplicaron buenas prácticas en la escritura de las pruebas unitarias para que fueran:

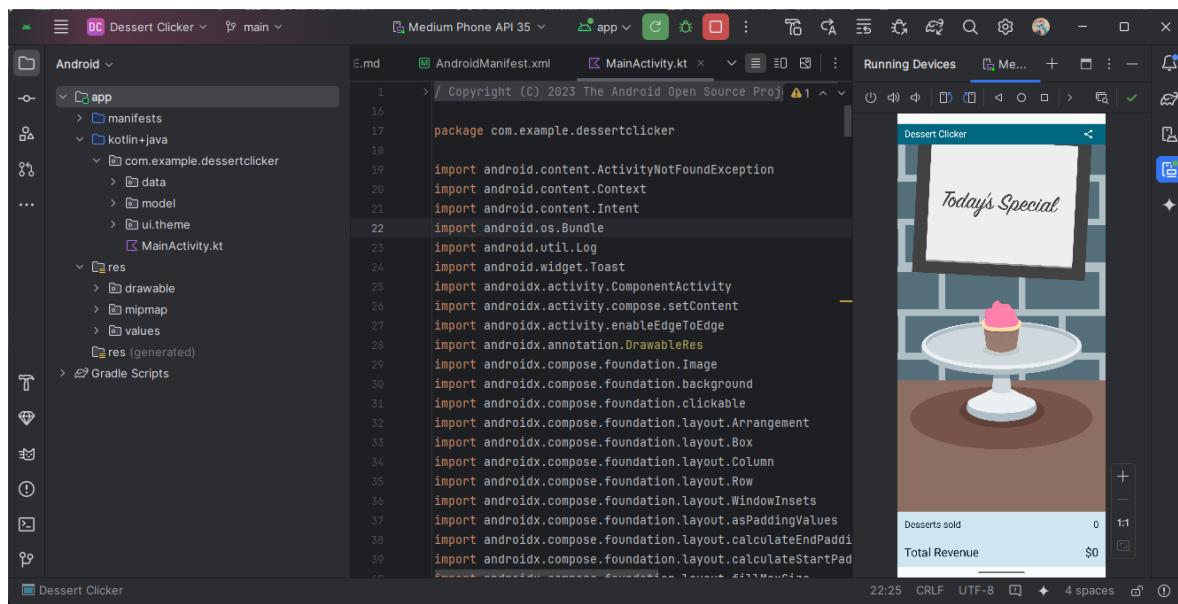
- **Enfocadas**, probando unidades individuales como métodos específicos del ViewModel.
- **Comprensibles**, con código claro y nombres de prueba descriptivos.
- **Deterministas**, con resultados consistentes en cada ejecución.
- **Independientes**, sin depender de interacción humana ni del estado de otras pruebas.

Estas pruebas no solo permiten validar el comportamiento previsto de la app, sino también detectar errores potenciales ante situaciones no contempladas durante el desarrollo inicial. Gracias a esta estrategia, se refuerza la confianza en la estabilidad del juego y se facilita su mantenimiento futuro.



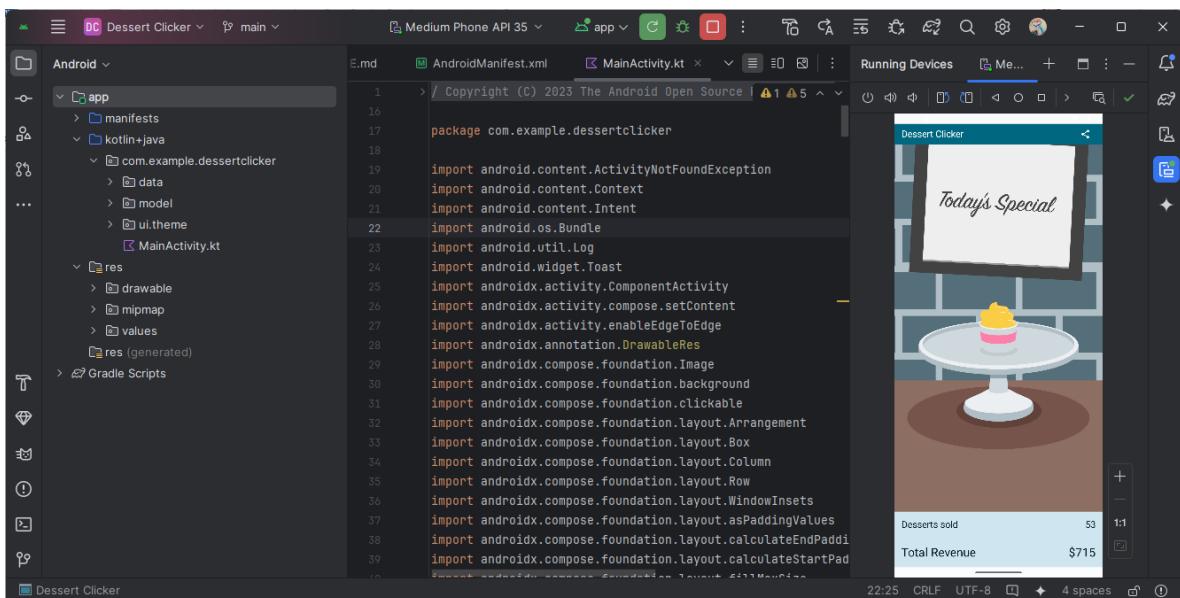
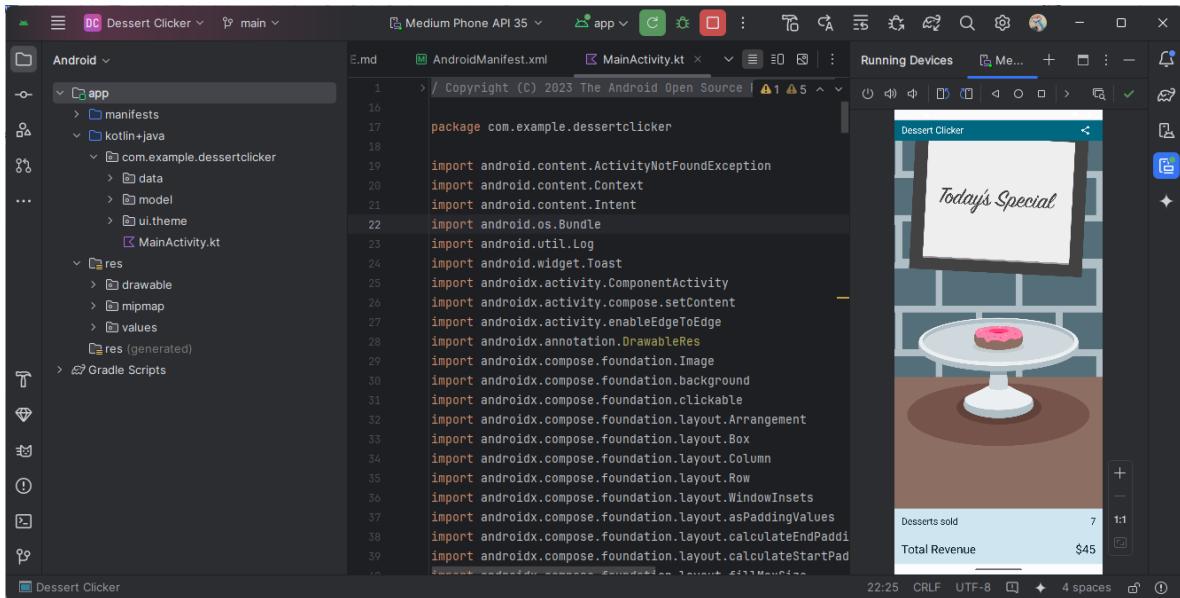
## Práctica: Agrega un ViewModel a Dessert Clicker

En el codelab de Dessert Clicker el proceso que se vio en las etapas del ciclo de vida mejora de la arquitectura de la aplicación que esta vez se centra en la adopción del patrón ViewModel para una gestión más eficiente y consciente del ciclo de vida de los datos de la interfaz de usuario. Inicialmente, la lógica y el estado que impulsan la UI residen directamente en la función de componibilidad principal dentro de MainActivity. El primer paso para la refactorización implica la configuración de la dependencia necesaria para el componente ViewModel de Jetpack Compose, lo que permite su integración en el proyecto. A continuación, se define una clase de datos específica para encapsular todo el estado relevante de la interfaz de usuario, proporcionando una estructura clara y organizada para la gestión de la información. El núcleo de la refactorización es la creación de una clase ViewModel, diseñada para almacenar y gestionar los datos de la UI de manera independiente del ciclo de vida de la actividad, sobreviviendo así a cambios de configuración como la rotación de pantalla. La lógica de negocio y la gestión del estado, que previamente residían en la capa de presentación (MainActivity), se trasladan al ViewModel, simplificando la responsabilidad de la actividad principal a la mera presentación de la UI. Finalmente, se establece la conexión entre la UI en MainActivity y el ViewModel, permitiendo que la interfaz de usuario observe el estado gestionado por el ViewModel y le delegue las acciones del usuario, lo que resulta en una arquitectura de aplicación más limpia, mantenible y robusta, con una clara separación entre la lógica de presentación y la lógica de negocio.



The screenshot shows the Android Studio interface with the 'Dessert Clicker' project open. The left sidebar displays the project structure under 'Android'. The main area shows the 'MainActivity.kt' file being edited. The code imports various AndroidX components and defines a class named 'MainActivity'. On the right, the 'Preview' tab of the design tool shows a mobile device screen with a cupcake on a stand and some UI statistics at the bottom.

```
Copyright (C) 2023 The Android Open Source Project
package com.example.dessertclicker
import android.content.ActivityNotFoundException
import android.content.Context
import android.content.Intent
import android.os.Bundle
import android.util.Log
import android.widget.Toast
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.activity.enableEdgeToEdge
import androidx.annotation.DrawableRes
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.WindowInsets
import androidx.compose.foundation.layout.asPaddingValues
import androidx.compose.foundation.layout.calculateEndPadding
import androidx.compose.foundation.layout.calculateStartPadding
```

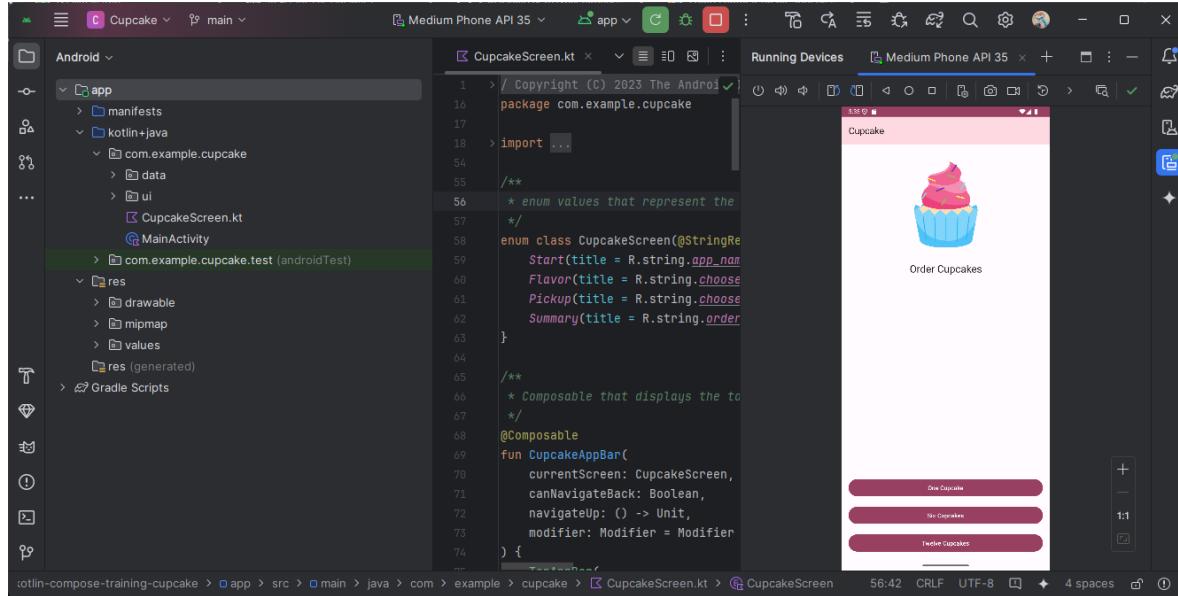


## Ruta de Aprendizaje 2 (Navigation en Jetpack Compose)

### Cómo navegar entre pantallas con Compose

Aquí se explica cómo el componente Navigation de Jetpack Compose facilita la definición de rutas únicas para cada pantalla y cómo asociar cada ruta con un elemento componible específico mediante el uso de un NavHost. El NavController se presenta como el responsable de gestionar la navegación entre estas rutas. En lugar de pasar directamente el NavController a las pantallas individuales, el codelab enfatiza la importancia de utilizar callbacks de eventos para desacoplar la lógica de navegación de la interfaz de usuario de cada pantalla, manteniendo la gestión de la navegación centralizada dentro del NavHost. Finalmente, se aborda la interacción con otras aplicaciones del sistema a través de Intents, mostrando cómo crear y

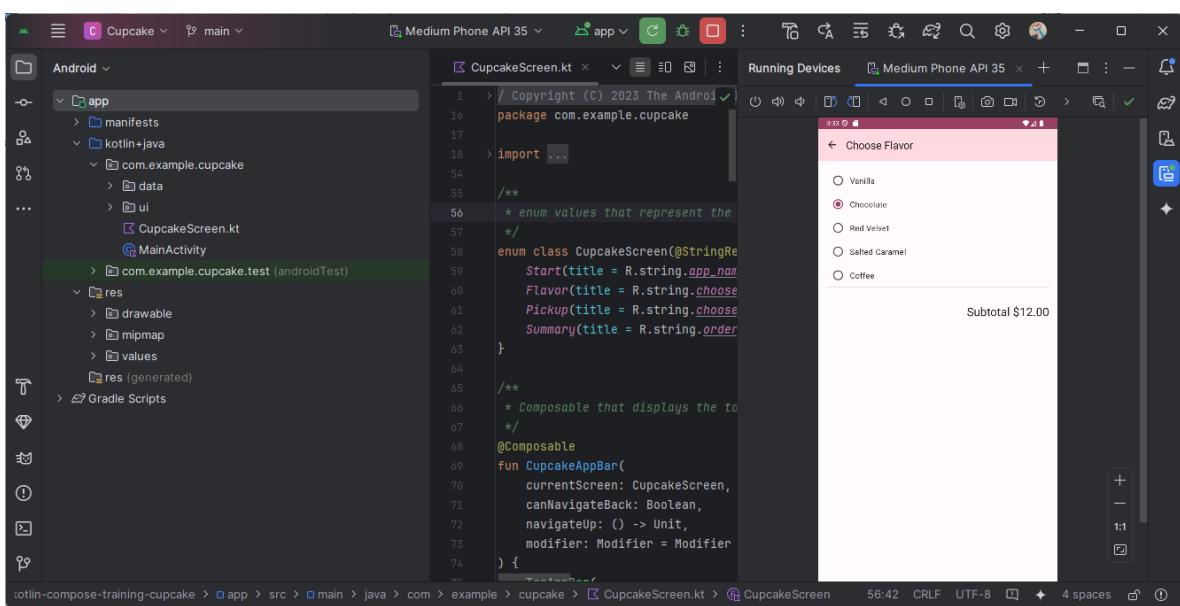
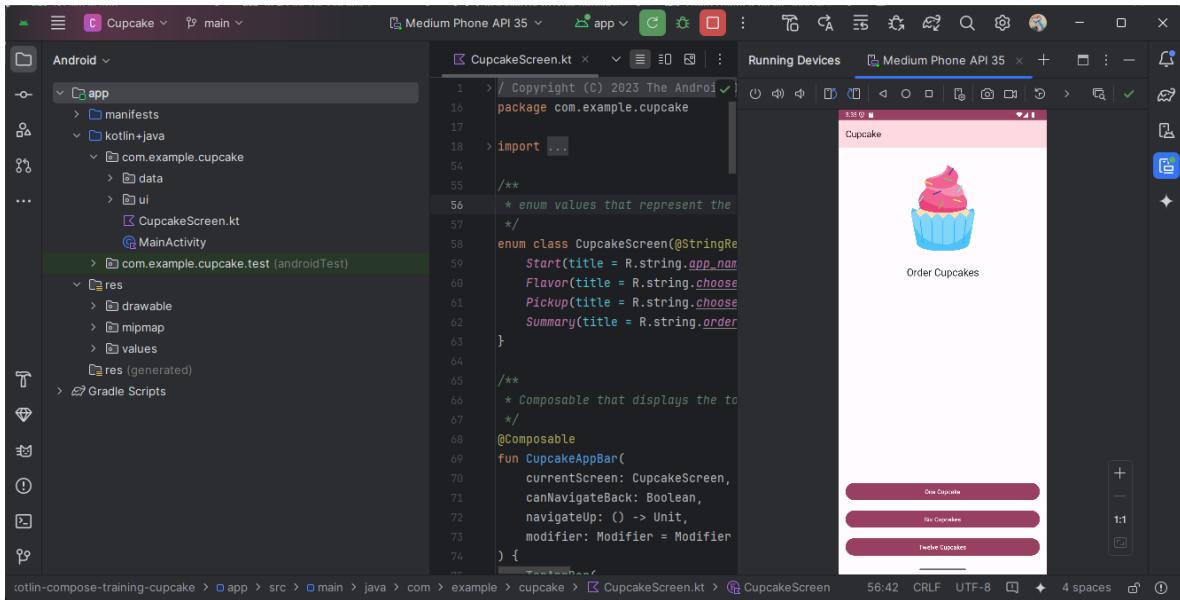
utilizar un Intent con la acción de compartir (ACTION\_SEND) para permitir a los usuarios enviar la información del pedido de cupcakes a otras aplicaciones, como las de mensajería o correo electrónico, utilizando la hoja de compartir nativa de Android.

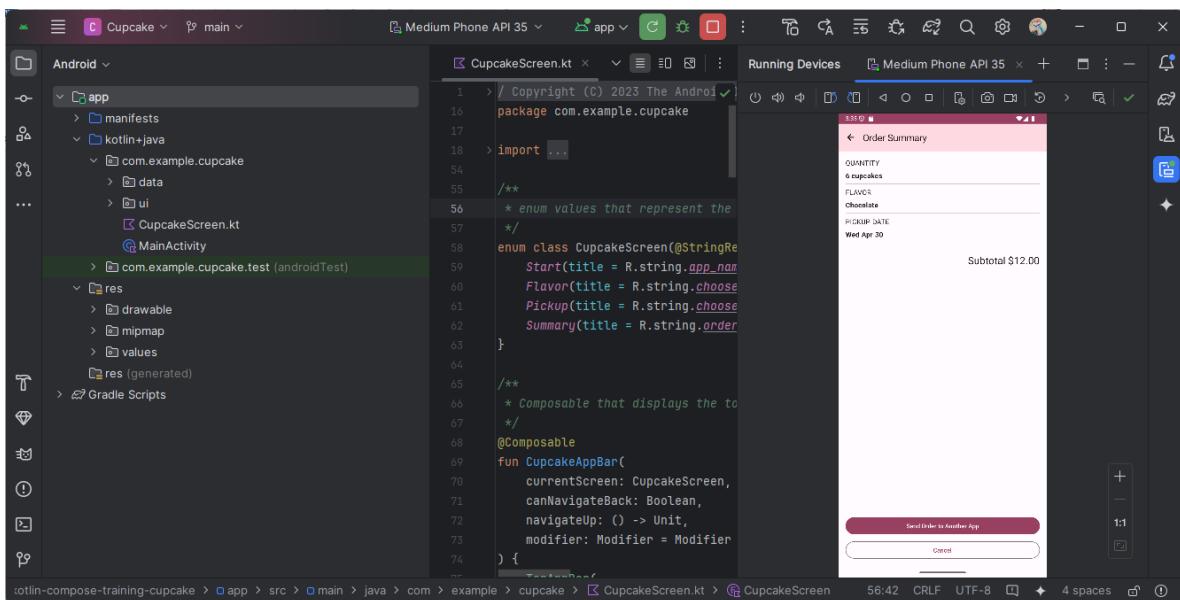
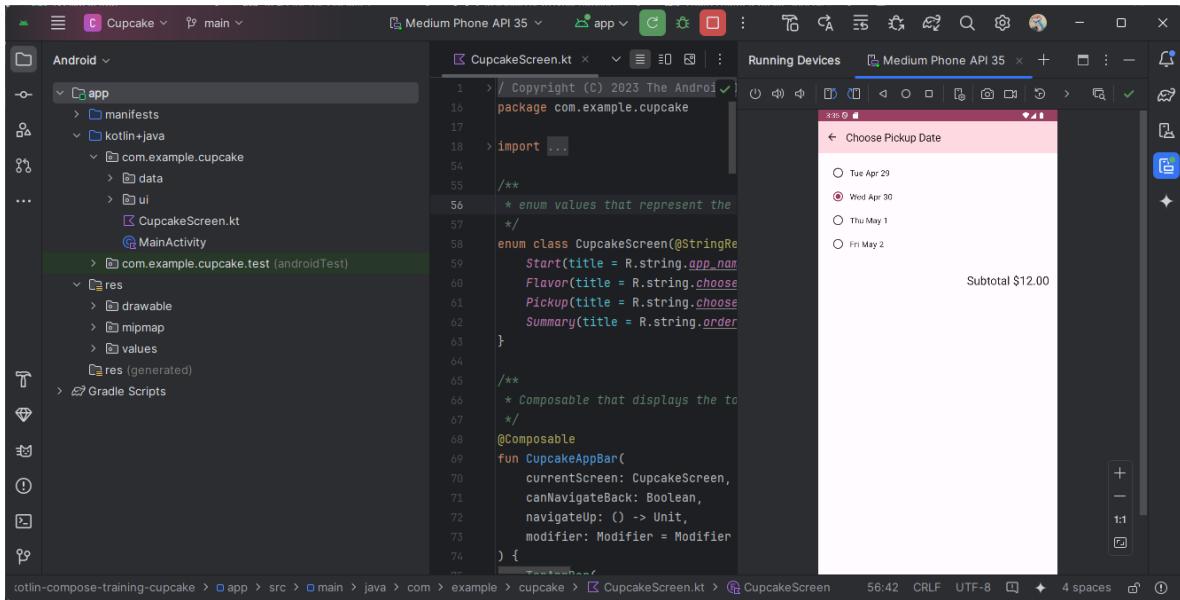


## Prueba la App de Cupcake

Inicialmente aquí se establece la configuración necesaria para las pruebas de navegación, lo que implica la creación y gestión de un TestNavController dentro de una AndroidComposeTestRule, permitiendo así la inspección del flujo de navegación de la aplicación. Posteriormente, el codelab guía en la implementación de pruebas específicas para verificar la correcta navegación entre las diferentes pantallas de la aplicación Cupcake, simulando las interacciones del usuario con los botones de navegación y asegurando que la aplicación se dirige a la pantalla esperada en cada caso, incluyendo la navegación hacia adelante, hacia atrás y la cancelación de flujos.

También se aborda la prueba del contenido de las pantallas individuales, utilizando la pantalla de selección de sabores como ejemplo para demostrar cómo verificar que los elementos de la UI, como las opciones de selección y los textos informativos, se presentan correctamente.

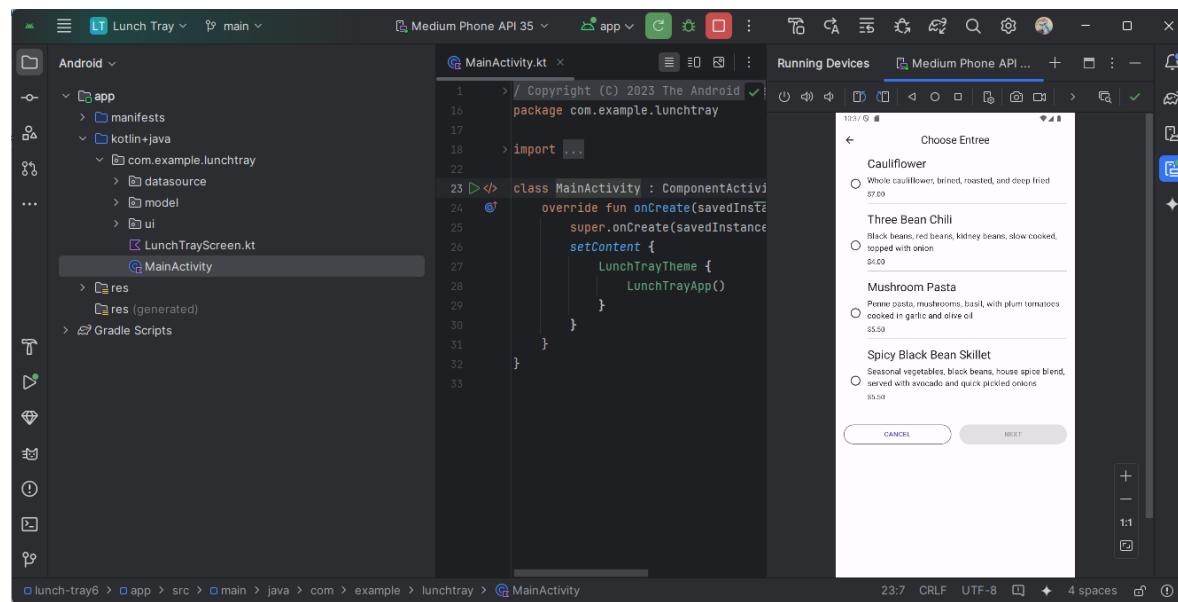
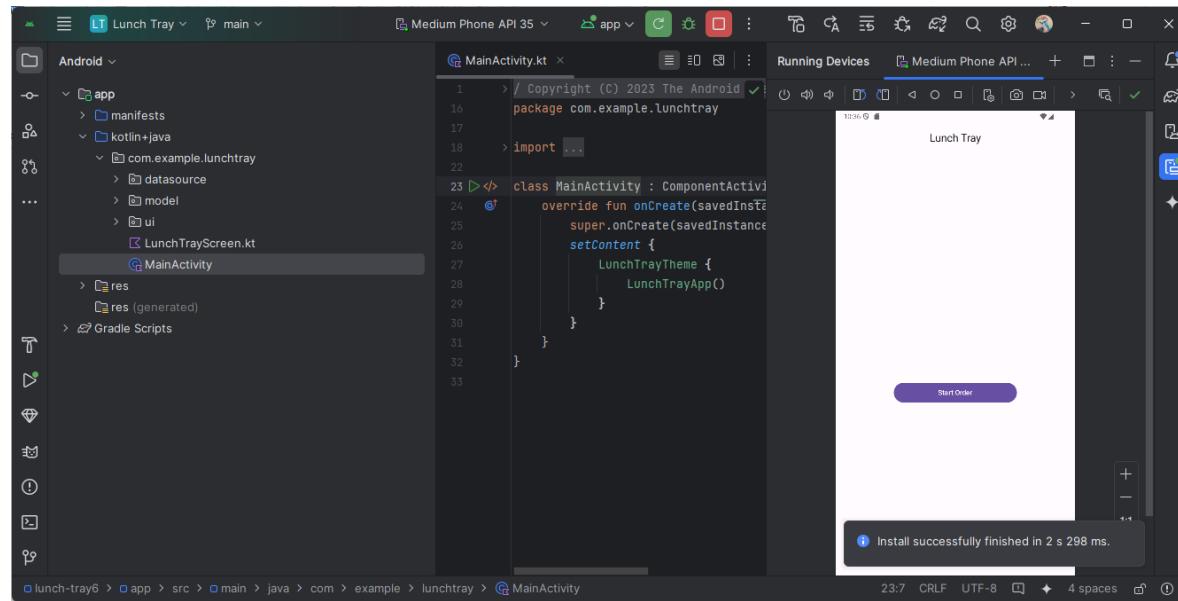


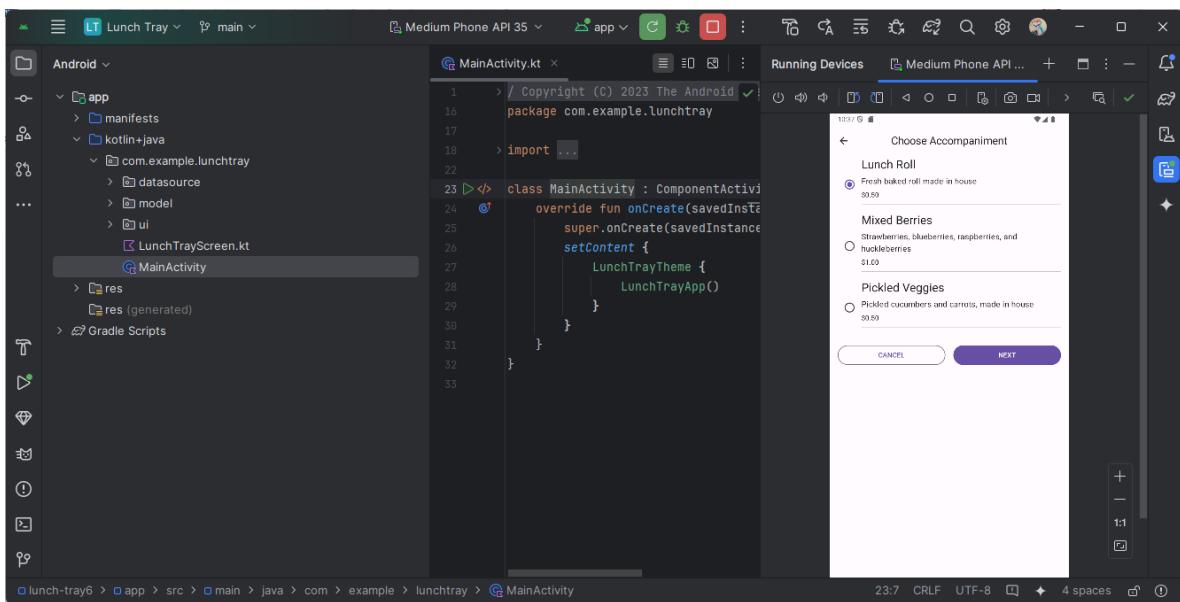
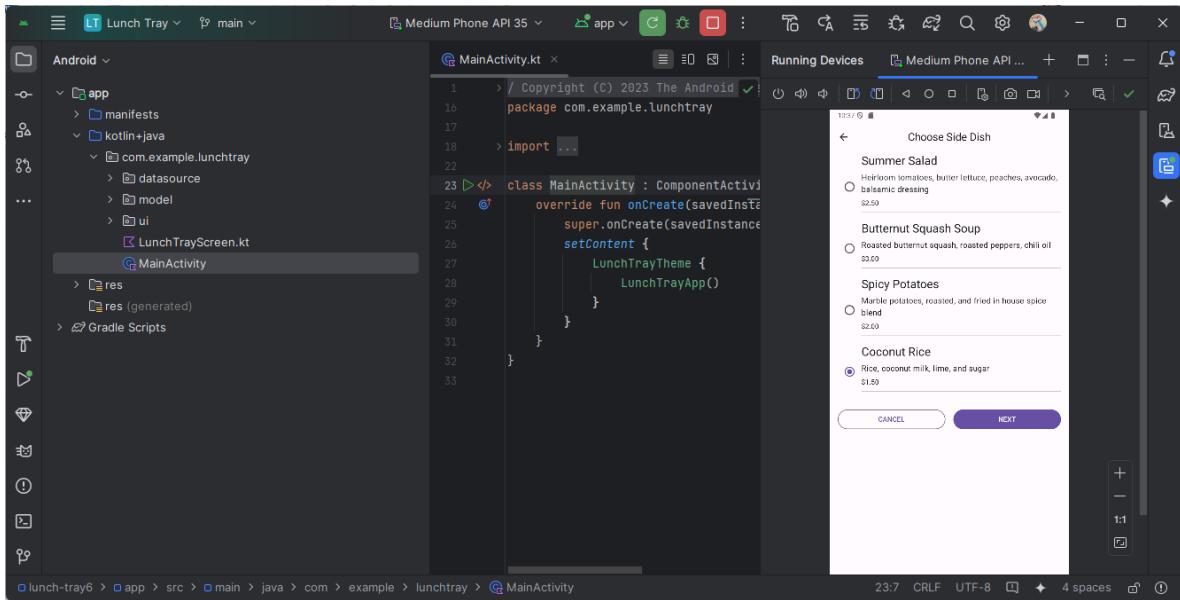


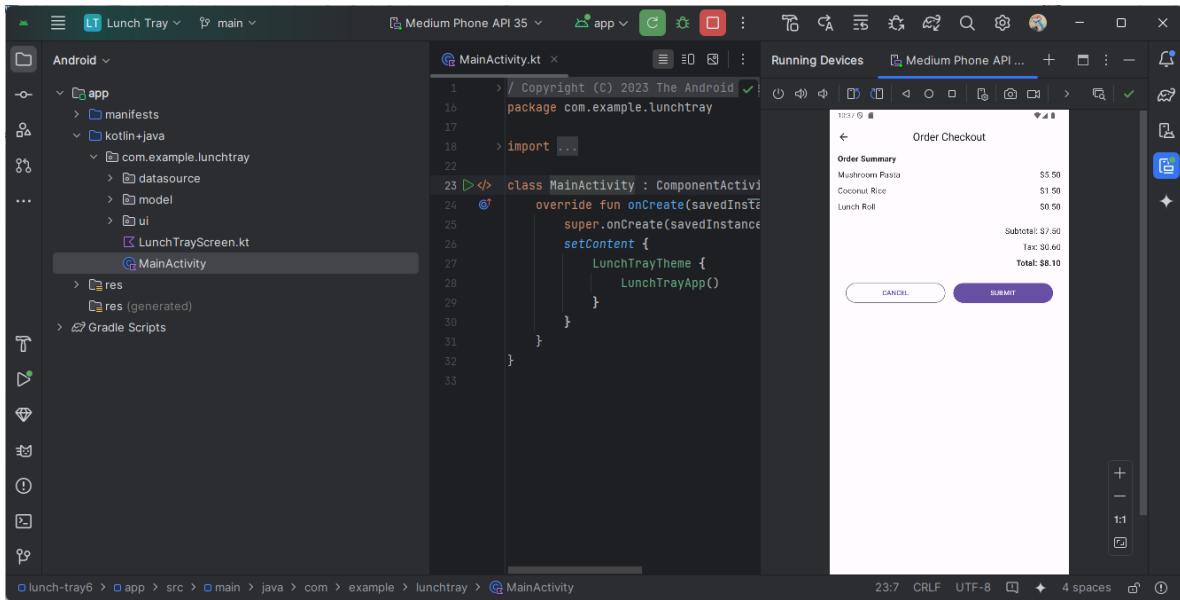
## Práctica: Cómo agregar navegación

Este codelab se centra en la aplicación de los conceptos de Navigation Compose para implementar un flujo de navegación completo en la aplicación Lunch Tray, que consta de varias pantallas interconectadas para la selección de elementos del almuerzo y la confirmación del pedido. El proceso comienza con la definición de una estructura clara de las pantallas a través de un enum, donde cada pantalla tiene un nombre y un título asociado. Además, se aborda la inicialización del NavController y la gestión del estado de la pantalla actual. Cabe mencionar que un componente crucial es la creación de una AppBar dinámica que muestra el título de la pantalla activa y un botón de retroceso contextual, que no debe aparecer en la pantalla de inicio. Finalmente, el núcleo del ejercicio reside en la configuración del NavHost,

donde se definen las rutas para cada pantalla y las transiciones entre ellas, basadas en las interacciones del usuario con los botones "Siguiente", "Enviar" y "Cancelar".



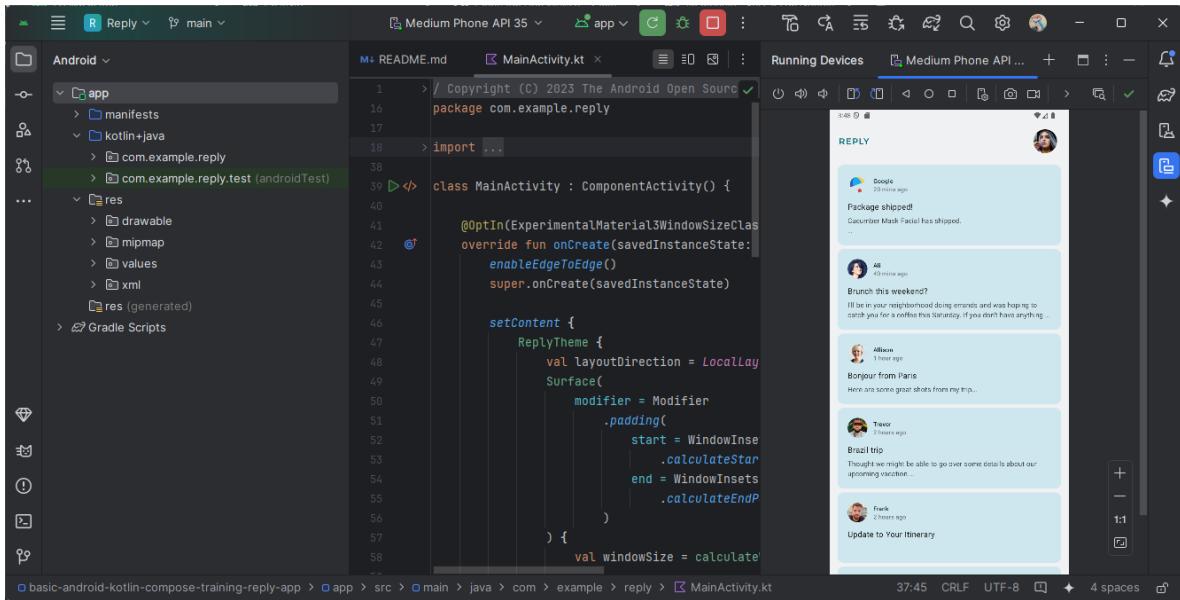




## Ruta de Aprendizaje 3 (Adáptate a diferentes tamaños de pantalla)

### Compila una app adaptable con navegación dinámica

El codelab de la aplicación Reply explora dos métodos principales para la navegación y la adaptabilidad del diseño en Jetpack Compose. Primero, se enseña cómo implementar una navegación sencilla entre dos pantallas (principal y detalles) utilizando el manejo de estados mutables en lugar de un NavController, y cómo crear un controlador de retroceso personalizado con BackHandler. Segundo, se centra en la creación de un diseño adaptable a diferentes tamaños de pantalla mediante la API de WindowSizeClass y la implementación de patrones de navegación de Material Design como la navegación inferior (para pantallas compactas), el riel de navegación (para pantallas medianas) y el panel lateral de navegación permanente (para pantallas expandidas), ajustando la interfaz de usuario para proporcionar una mejor experiencia en diversos dispositivos.



## Compila una app adaptable con un diseño adaptable

Este codelab se centra en adaptar la app de Reply que vimos anteriormente para pantallas grandes implementando el patrón de vista de lista-detalles. Se aprende a crear vistas previas específicas para diferentes tamaños de pantalla y a usar la WindowSizeClass para determinar si mostrar solo la lista de correos o la lista junto con los detalles. Se ajustan los componentes de la interfaz de usuario de la pantalla de detalles para que se integren mejor en el diseño de lista-detalles y se modifica el comportamiento del botón de retroceso para cerrar la app en pantallas grandes en lugar de navegar a otra pantalla. El objetivo es optimizar la experiencia del usuario en tablets y dispositivos con pantallas grandes mostrando más información simultáneamente.

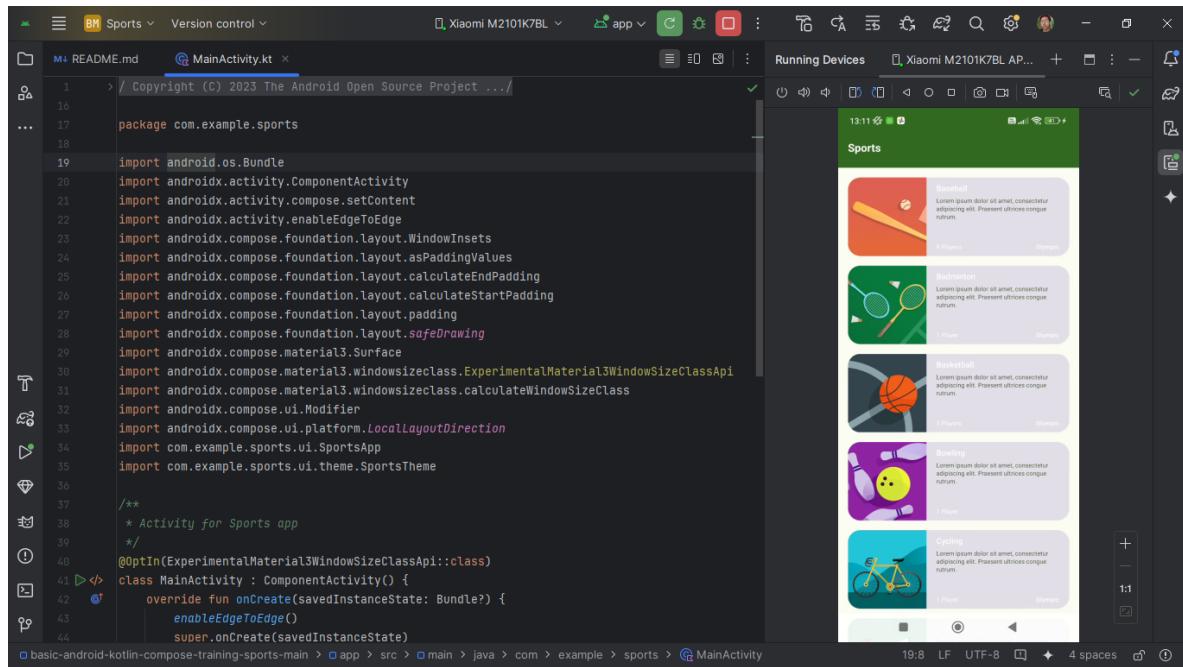
### Práctica: Compila una app de deportes

Aquí aprendemos sobre la adaptación (de la aplicación Sports en este caso) para pantallas grandes que comienza con la fase de planificación, donde se analiza el diseño actual de la aplicación en pantallas compactas, que consta de una lista y una pantalla de detalles separada.

La siguiente etapa se centra en la implementación de este diseño expandido en el código. Se guía al usuario para crear un nuevo elemento componible llamado SportsListAndDetails, destinado a mostrar la lista de deportes y los detalles de un deporte en la misma pantalla. Se destaca la importancia de crear una vista previa para este nuevo componente y de configurar correctamente el comportamiento del botón "Atrás" en este contexto, específicamente para que cierre la aplicación cuando se presiona desde la vista principal expandida.

Finalmente, el codelab aborda la lógica para que la aplicación cambie dinámicamente su diseño según el tamaño de la pantalla. Esto implica la integración

de la biblioteca material3-window-size-class, el cálculo de la clase de tamaño de ventana en la actividad principal y la creación de un mecanismo para determinar el tipo de contenido (ListOnly o ListAndDetail) basado en el ancho de la ventana.



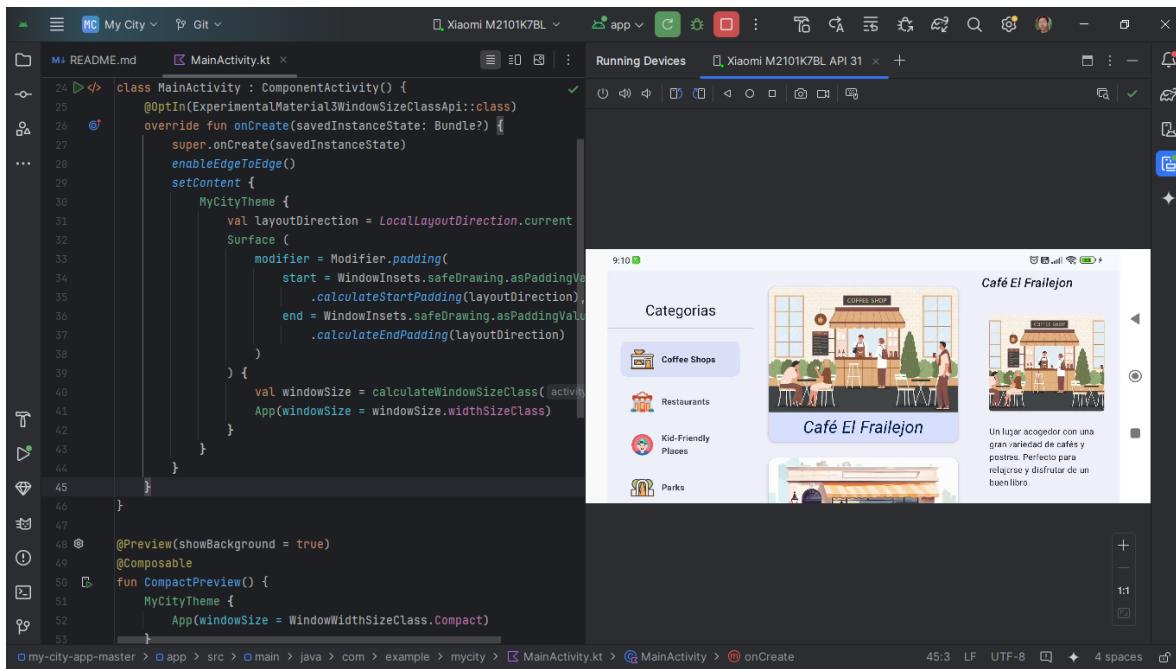
### Proyecto: Crea una app llamada My city

Para este proyecto, se desarrolló una aplicación llamada Mi City, la cual tiene como objetivo mostrar recomendaciones de actividades y lugares para visitar en una ciudad elegida por nosotros. Esta aplicación fue parte de la Unidad 4 del curso, y nos permitió aplicar de forma práctica todo lo aprendido sobre el desarrollo de aplicaciones con varias pantallas, diseño adaptable y arquitectura moderna en Android usando Jetpack Compose.

Durante el desarrollo, se implementaron varias pantallas que representan distintas categorías de recomendaciones, como cafeterías, parques y lugares para salir con niños. Se hizo uso del componente Navigation de Jetpack para manejar la navegación entre estas pantallas de manera ordenada y clara. Además, se mantuvo una separación entre la capa de interfaz de usuario y los datos, utilizando ViewModels y el patrón de flujo unidireccional de datos para actualizar la IU según el estado actual.

También pudimos aprender cómo adaptar el diseño para que funcione correctamente en distintos tamaños de pantalla, y cómo respetar el ciclo de vida de la actividad para evitar desperdiciar recursos. Este proyecto nos ayudó a consolidar lo aprendido, especialmente al enfrentarnos a decisiones de diseño, resolución de errores y organización del contenido.

Más allá de seguir los pasos del codelab, esta aplicación nos retó a aplicar el conocimiento en un contexto nuevo, lo que nos dio más seguridad para resolver problemas como lo haría un desarrollador en el mundo real. En resumen, este proyecto fue una excelente oportunidad para reforzar nuestras habilidades en Compose y en la arquitectura de apps modernas.



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

- Left Panel (Project Structure):** Shows the project structure with files like README.md and MainActivity.kt.
- Code Editor (MainActivity.kt):** Displays the Kotlin code for the main activity. The code includes logic for calculating padding based on window insets and window size classes, and a preview of the app's UI.
- Preview Window:** Shows a preview of the app's UI on a "Xiaomi M2101K7BL API 31" device. The UI features a navigation bar with categories: "Cafes", "Restaurants", "Kid-Friendly Places", and "Parks". A specific card for "Café El Frailejon" is displayed, showing a small illustration of a coffee shop and some descriptive text.
- Bottom Status Bar:** Shows the file path "my-city-app-master > app > src > main > java > com > example > mycity > MainActivity.kt", the line number "45", and other development settings.

**Nota:** Los códigos con comentarios se encuentran subidos en [github](#).

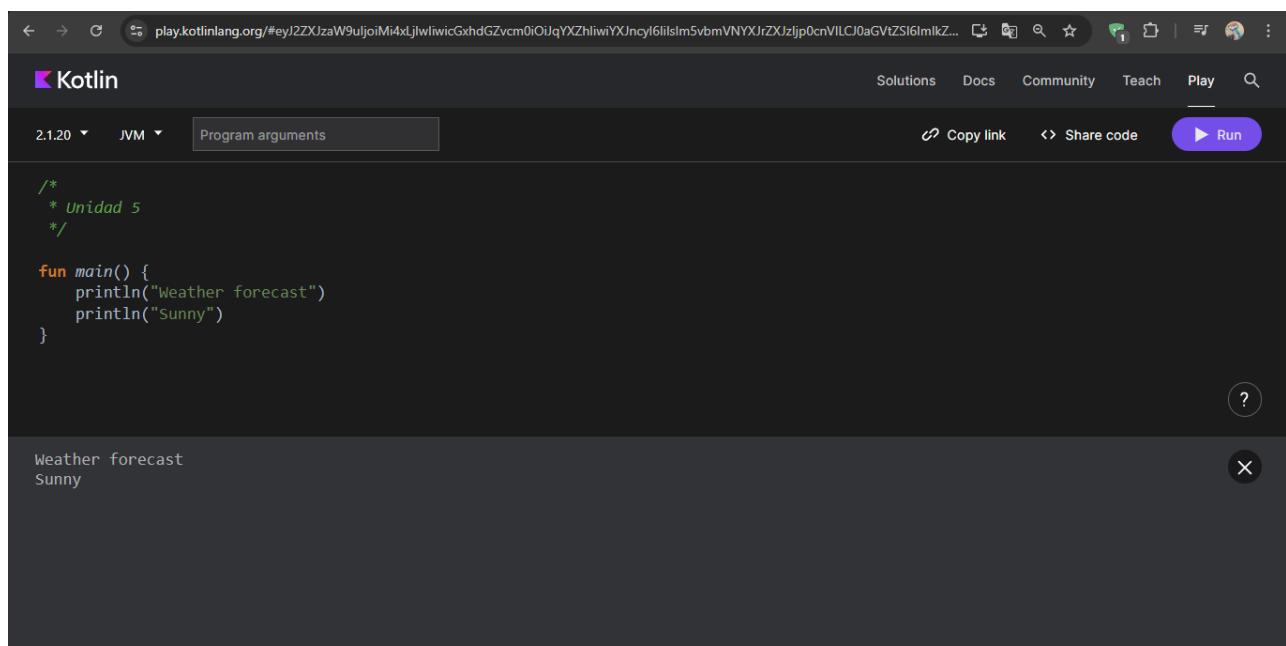
## Unidad 5: Cómo conectarse a Internet

### Ruta de Aprendizaje 1 (Cómo obtener datos de Internet)

#### Introducción a las corrutinas en el Playground de Kotlin

Este codelab de Android tiene como objetivo el enseñar cómo mejorar el rendimiento de una aplicación utilizando programación asíncrona y concurrente. El enfoque inicial compara una ejecución secuencial básica sin corrutinas con otra que utiliza `delay()` para simular un retraso, mostrando cómo el uso de `runBlocking` permite introducir esta pausa sin necesidad de bloquear completamente el hilo principal.

En este caso este código de Kotlin define una función principal (`main`) que contiene dos instrucciones. La primera instrucción utiliza la función `println()` para mostrar la frase "Weather forecast" en la consola. Acto seguido, la segunda instrucción también emplea `println()` para imprimir la palabra "Sunny" en la línea siguiente. En esencia, el programa toma estas dos cadenas de texto y las presenta secuencialmente en la salida del programa, cada una en una línea separada.



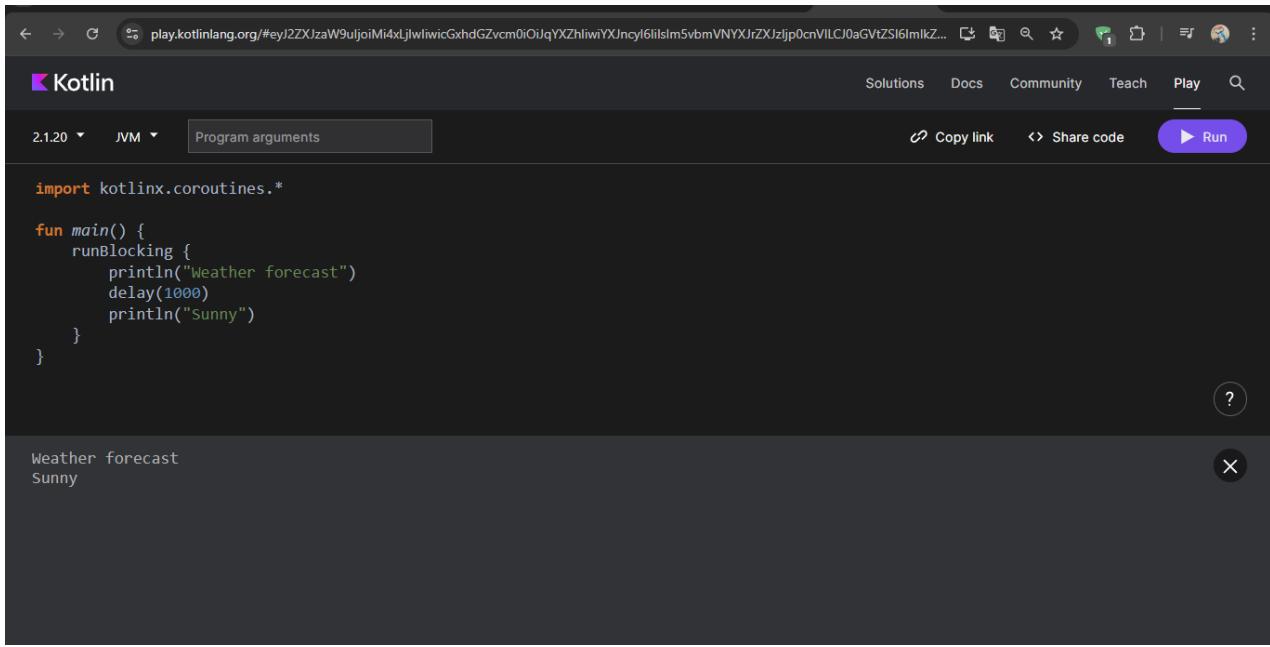
The screenshot shows the Kotlin Playground interface. The code area contains the following Kotlin code:

```
/*
 * Unidad 5
 */

fun main() {
    println("Weather forecast")
    println("Sunny")
}
```

The "Run" button is highlighted in purple. Below the code, the terminal window displays the output:

```
Weather forecast
Sunny
```



A screenshot of the Kotlin Play IDE interface. The top navigation bar includes links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the bar, there are dropdown menus for '2.1.20' and 'JVM'. A 'Program arguments' input field is present. On the right, there are buttons for 'Copy link', 'Share code', and 'Run'. The main area displays the following Kotlin code:

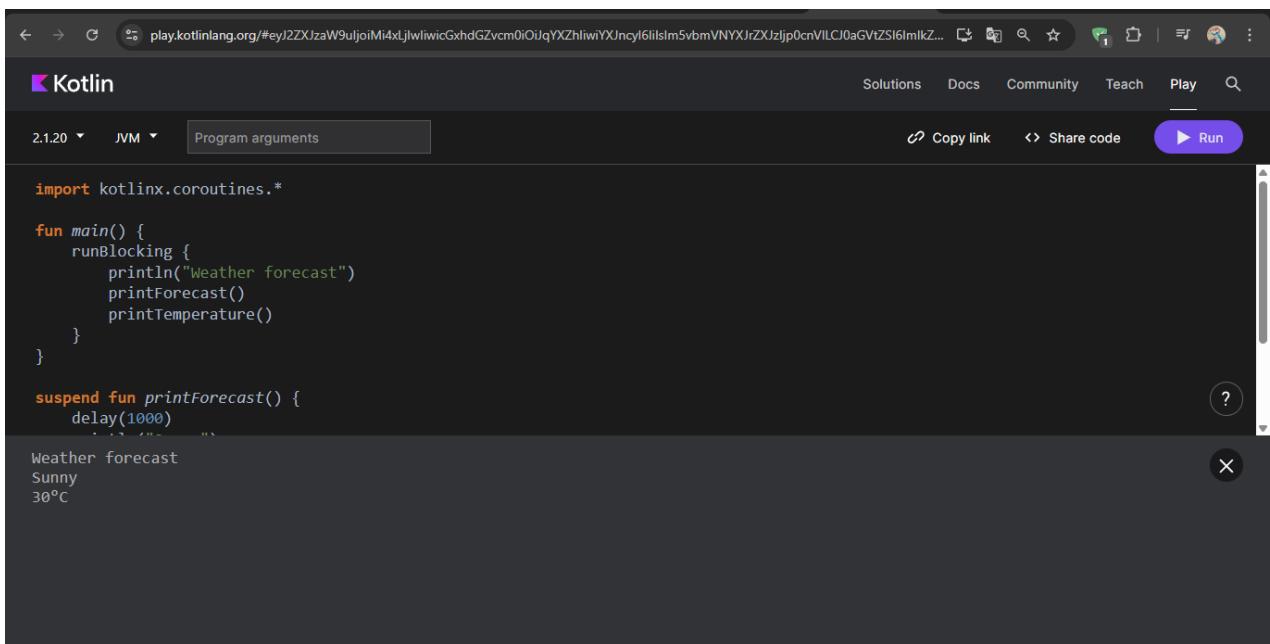
```
import kotlinx.coroutines.*

fun main() {
    runBlocking {
        println("Weather forecast")
        delay(1000)
        println("Sunny")
    }
}
```

The output window shows the results of the execution:

```
Weather forecast
Sunny
```

Aquí podemos ver cómo se está imprimiendo "Weather forecast". Acto seguido, se invoca una función suspendida llamada printForecast(), la cual introduce una pausa de un segundo, antes de imprimir "Sunny". Simultáneamente o justo después, se llama a la función printTemperature() que imprime "30°C". La función runBlocking asegura que el programa principal espere a que tanto la impresión de "Sunny" (con su retardo) como la impresión de la temperatura se completen antes de finalizar su ejecución. En resumen, el programa simula una pequeña espera antes de proporcionar la parte del pronóstico del tiempo.



A screenshot of the Kotlin Play IDE interface, identical to the first one but with different code and output. The top navigation bar, code editor, and output window are all visible. The code is as follows:

```
import kotlinx.coroutines.*

fun main() {
    runBlocking {
        println("Weather forecast")
        printForecast()
        printTemperature()
    }
}

suspend fun printForecast() {
    delay(1000)
}
```

The output window shows the results of the execution:

```
Weather forecast
Sunny
30°C
```

The screenshot shows the Kotlin Play IDE interface. The code editor contains the following Kotlin code:

```
2.1.20 JVM Program arguments
    }
    println("Execution time: ${time / 1000.0} seconds")
}
suspend fun printForecast() {
    delay(1000)
    println("Sunny")
}

suspend fun printTemperature() {
    delay(1000)
    println("30\u00b0C")
}

Weather forecast
Sunny
30°C
Execution time: 2.106 seconds
```

The output window at the bottom displays the results of the execution: "Weather forecast", "Sunny", "30°C", and "Execution time: 2.106 seconds".

Además, también se introducen las funciones suspendidas (suspend fun), que permiten que una función se ejecute de manera asíncrona. Estas funciones, como printForecast y printTemperature, permiten simular tareas que se suspenden y reanudan de forma eficiente. Luego, se incorpora measureTimeMillis para medir cuánto tarda en ejecutarse un bloque de código, evidenciando las diferencias de rendimiento entre ejecución secuencial y paralela.

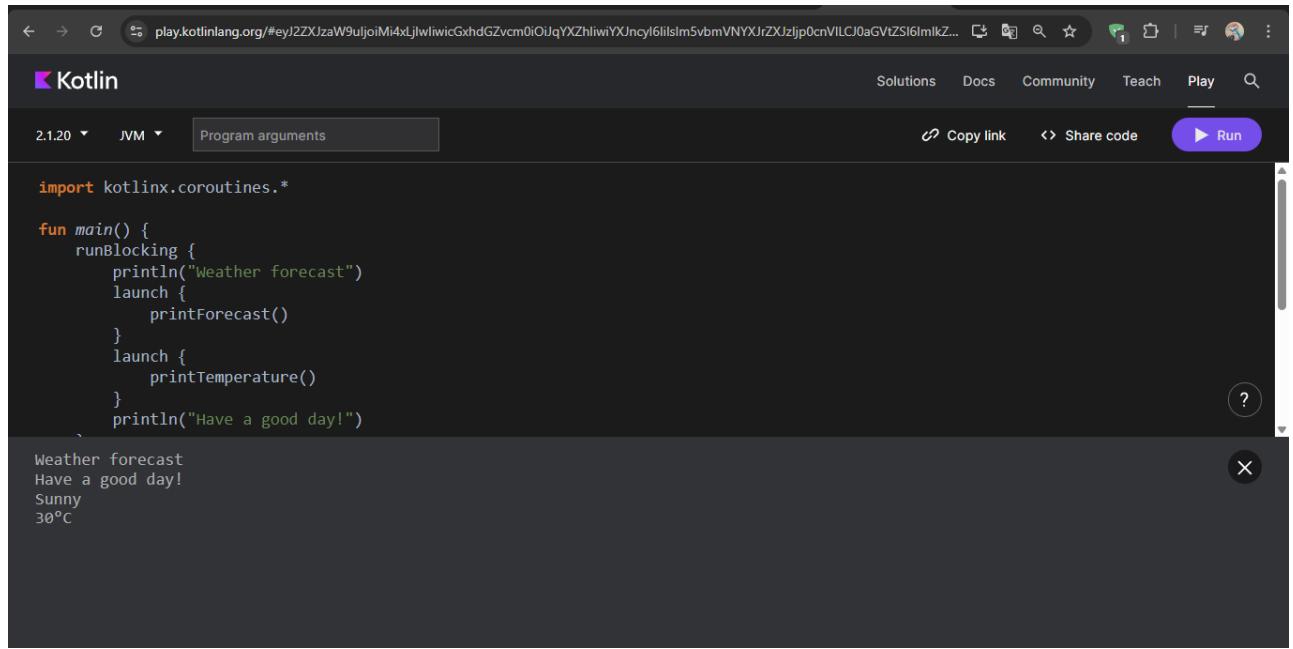
The screenshot shows the Kotlin Play IDE interface. The code editor contains the following Kotlin code:

```
import kotlinx.coroutines.*

fun main() {
    runBlocking {
        println("weather forecast")
        launch {
            printForecast()
        }
        launch {
            printTemperature()
        }
    }
}

Weather forecast
Sunny
30°C
```

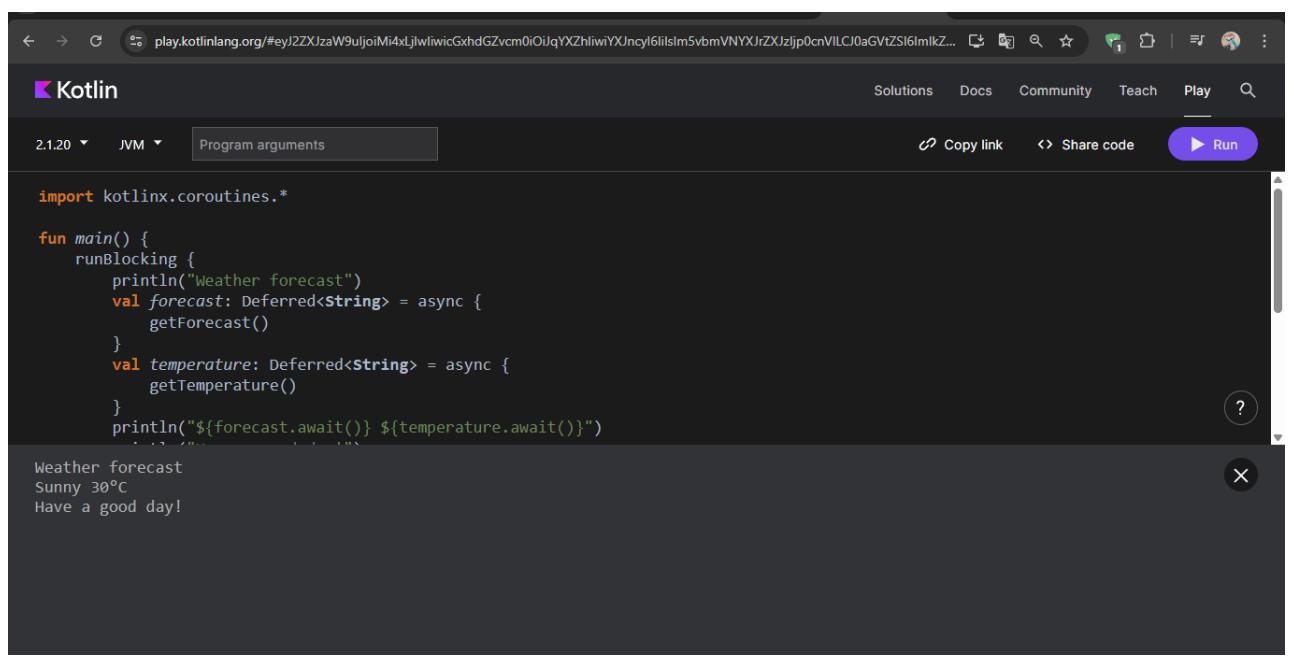
The output window at the bottom displays the results of the execution: "Weather forecast", "Sunny", and "30°C".



The screenshot shows the Kotlin Play IDE interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, and Play. Below the navigation bar, there are dropdown menus for '2.1.20' and 'JVM'. A 'Program arguments' input field is present. On the right side, there are buttons for 'Copy link', 'Share code', and a purple 'Run' button. The main area displays the following Kotlin code:

```
import kotlinx.coroutines.*  
  
fun main() {  
    runBlocking {  
        println("Weather forecast")  
        launch {  
            printForecast()  
        }  
        launch {  
            printTemperature()  
        }  
        println("Have a good day!")  
    }  
}  
  
Weather forecast  
Have a good day!  
Sunny  
30°C
```

The output window below the code shows the execution results: "Weather forecast", "Have a good day!", "Sunny", and "30°C".



This screenshot shows another instance of the Kotlin Play IDE. The interface is identical to the first one, with the same navigation bar, dropdowns, and run buttons. The main area contains the following Kotlin code:

```
import kotlinx.coroutines.*  
  
fun main() {  
    runBlocking {  
        println("Weather forecast")  
        val forecast: Deferred<String> = async {  
            getForecast()  
        }  
        val temperature: Deferred<String> = async {  
            getTemperature()  
        }  
        println("${forecast.await()} ${temperature.await()}")  
    }  
}  
  
Weather forecast  
Sunny 30°C  
Have a good day!
```

The output window shows the combined results: "Weather forecast", "Sunny 30°C", and "Have a good day!".

El codelab continúa mostrando cómo lanzar corutinas de forma concurrente usando `launch` para que varias tareas puedan ejecutarse al mismo tiempo. Posteriormente, se introduce `async` junto con `await`, para lanzar tareas que retornan un resultado y luego combinar esos resultados de forma eficiente. Esta técnica permite construir funciones compuestas como `getWeatherReport`, que paraleliza subtareas como `getForecast` y `getTemperature` dentro de un `coroutineScope`.

The screenshot shows the Kotlin Play IDE interface. The code editor contains the following Kotlin code:

```
import kotlinx.coroutines.*

fun main() {
    runBlocking {
        println("Weather forecast")
        println(getWeatherReport())
        println("Have a good day!")
    }
}

suspend fun getWeatherReport() = coroutineScope {
    val forecast = async { getForecast() }
    // ... (code omitted)
}
```

The output window displays the results of the execution:

```
Weather forecast
Sunny 30°C
Have a good day!
```

The screenshot shows the Kotlin Play IDE interface. The code editor contains the following Kotlin code:

```
fun main() {
    val numberOfPeople = 0
    val numberOfPizzas = 20
    println("slices per person: ${numberOfPizzas / numberOfPeople}")
}
```

The output window displays the exception message:

```
Exception in thread "main" java.lang.ArithmetricException: / by zero
at FileKt.main (File.kt:4)
at FileKt.main (File.kt:-1)
at jdk.internal.reflect.NativeMethodAccessorImpl.invoke0 (:-2)
```

Kotlin

Solutions Docs Community Teach Play

2.1.20 JVM Program arguments

```
import kotlinx.coroutines.*
```

```
fun main() {
    runBlocking {
        println("Weather forecast")
        println(getWeatherReport())
        println("Have a good day!")
    }
}
```

```
suspend fun getWeatherReport() = coroutineScope {
    val forecast = async { getForecast() }
```

Weather forecast

Exception in thread "main" java.lang.AssertionError: Temperature is invalid  
at Filekt.getTemperature (File.kt:24)  
at Filekt\$getTemperature\$1.invokeSuspend (File.kt:-1)  
at kotlin.coroutines.jvm.internal.BaseContinuationImpl.resumeWith (ContinuationImpl.kt:33)

Kotlin

Solutions Docs Community Teach Play

2.1.20 JVM Program arguments

```
import kotlinx.coroutines.*
```

```
fun main() {
    runBlocking {
        println("Weather forecast")
        try {
            println(getWeatherReport())
        } catch (e: AssertionError) {
            println("Caught exception in runBlocking(): $e")
            println("Report unavailable at this time")
        }
        println("Have a good day!")
    }
}
```

Weather forecast

Caught exception in runBlocking(): java.lang.AssertionError: Temperature is invalid  
Report unavailable at this time  
Have a good day!

The screenshot shows the Kotlin Play IDE interface. The code editor contains the following Kotlin code:

```
2.1.20 JVM Program arguments
}
suspend fun getForecast(): String {
    delay(1000)
    return "Sunny"
}

suspend fun getTemperature(): String {
    delay(500)
    throw AssertionError("Temperature is invalid")
}
}

Weather forecast
Caught exception java.lang.AssertionError: Temperature is invalid
Sunny { No temperature found }
Have a good day!
```

The output window below the code editor displays the following text:

Weather forecast  
Caught exception java.lang.AssertionError: Temperature is invalid  
Sunny { No temperature found }  
Have a good day!

The screenshot shows the Kotlin Play IDE interface. The code editor contains the following Kotlin code:

```
2.1.20 JVM Program arguments
val temperature = async { getTemperature() }

delay(200)
temperature.cancel()

"${forecast.await()}"
}

suspend fun getForecast(): String {
    delay(1000)
    return "Sunny"
}

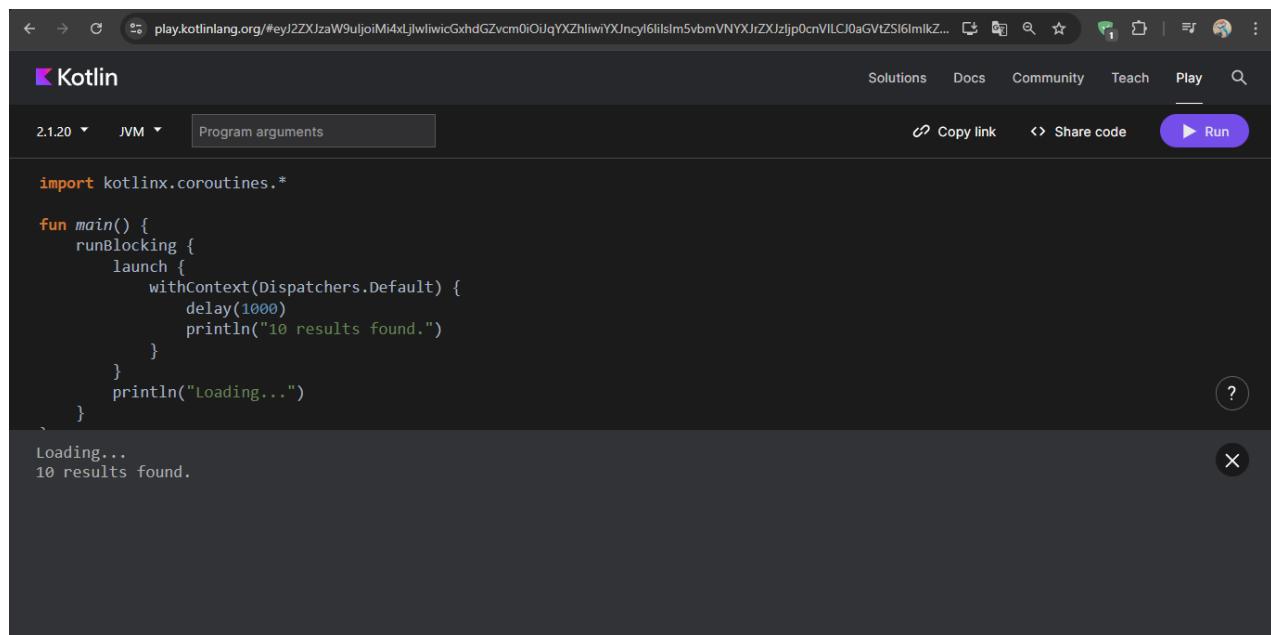
Weather forecast
Sunny
Have a good day!
```

The output window below the code editor displays the following text:

Weather forecast  
Sunny  
Have a good day!

En este caso se vuelve a hacer uso de corrutinas para ejecutar tareas de manera asíncrona, donde dentro de la función principal (main), se inicia un bloque runBlocking que espera la finalización de todas las corrutinas que se lancen dentro de él. Inmediatamente, se lanza una nueva corriputina utilizando launch. Dentro de esta corriputina, se cambia el contexto de ejecución a un grupo de hilos predeterminado (Dispatchers.Default) utilizando withContext. En este contexto, se introduce una pausa de un segundo (delay(1000)), simulando una operación que toma tiempo. Una vez transcurrido este segundo, se imprime el mensaje "10 results found.". Después de lanzar esta corriputina (pero antes de que termine debido al

delay), se ejecuta la instrucción `println("Loading...")` dentro del bloque `runBlocking`. Esto significa que "Loading..." se imprimirá casi inmediatamente después de que se inicie la corutina, y "10 results found." se imprimirá después de la pausa de un segundo. Se simula una carga de datos mostrando un mensaje de "Cargando..." y luego, tras una breve espera, revela que se han encontrado 10 resultados.



```
import kotlinx.coroutines.*  
  
fun main() {  
    runBlocking {  
        launch {  
            withContext(Dispatchers.Default) {  
                delay(1000)  
                println("10 results found.")  
            }  
        }  
        println("Loading...")  
    }  
}  
  
Loading...  
10 results found.
```

Por último, en este programa de Kotlin se explora el lanzamiento y la ejecución de corrutinas. Inicialmente, dentro de un bloque `runBlocking` que espera la finalización de todas las corrutinas internas, se imprime un mensaje indicando que se está ejecutando en la función `runBlocking` y el nombre del hilo principal. Luego, se lanza una nueva corutina con `launch`, dentro de la cual se imprime un mensaje con el nombre del hilo de la corutina. Dentro de esta corutina lanzada, se utiliza `withContext(Dispatchers.Default)` para cambiar el contexto de ejecución a un grupo de hilos optimizado para tareas intensivas, donde también se imprime el nombre del hilo y se introduce una pausa de un segundo. Después de la pausa, se imprime un mensaje indicando que se encontraron "10 resultados". Finalmente, una vez que la corutina lanzada se completa, se imprime un mensaje marcando el final de su ejecución. El código demuestra cómo se pueden crear y gestionar corrutinas para ejecutar tareas de forma concurrente y cómo se puede controlar el hilo en el que se ejecutan estas tareas.

The screenshot shows the Kotlin Play IDE interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, Play, and a search icon. Below the navigation bar is a toolbar with dropdowns for '2.1.20' and 'JVM', a 'Program arguments' input field, and buttons for 'Copy link', 'Share code', and 'Run'. The main area contains a code editor with the following Kotlin code:

```
import kotlinx.coroutines.*  
  
fun main() {  
    runBlocking {  
        println("${Thread.currentThread().name} - runBlocking function")  
        launch {  
            println("${Thread.currentThread().name} - launch function")  
            withContext(Dispatchers.Default) {  
                println("${Thread.currentThread().name} - withContext function")  
                delay(1000)  
                println("10 results found.")  
            }  
        }  
    }  
    main @coroutine#1 - runBlocking function  
    main @coroutine#1 - Loading...  
    main @coroutine#2 - launch function  
    DefaultDispatcher-worker-1 @coroutine#2 - withContext function  
    10 results found.  
    main @coroutine#2 - end of launch function
```

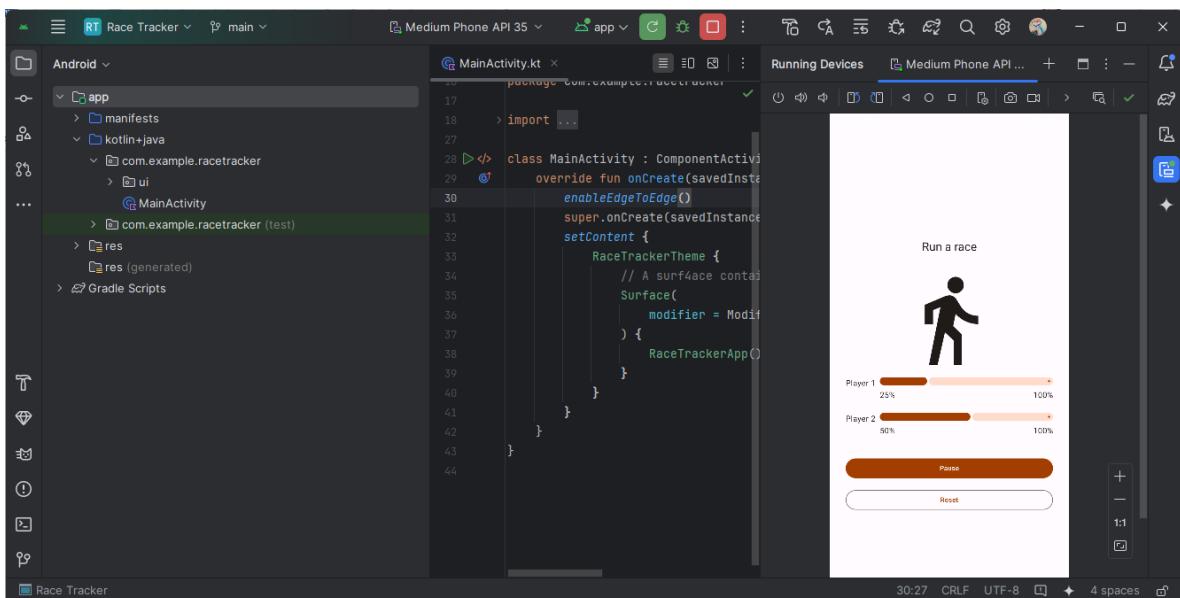
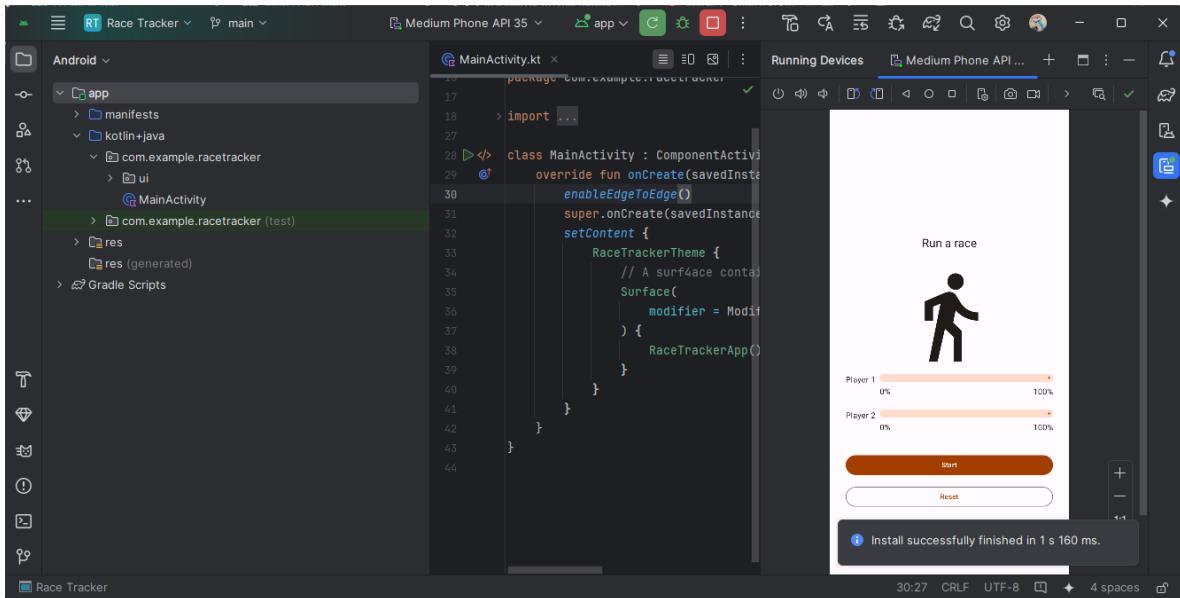
Below the code editor, the output window displays the execution results of the code.

## Introducción a las corrutinas en Android Studio

En este codelab se simula una carrera entre dos jugadores. La aplicación cuenta con una interfaz de usuario básica que incluye botones de inicio/pausa y reinicio, junto con barras de progreso para cada corredor. El objetivo principal del ejercicio es aprender a utilizar corrutinas para que ambos jugadores simulen correr al mismo tiempo, sin bloquear la interfaz de usuario, lo que permite que las barras de progreso se actualicen de manera fluida durante la simulación.

Se utiliza la clase `RaceParticipant` para definir el comportamiento de cada corredor, incluyendo su velocidad simulada mediante pausas y el incremento de su progreso. La función clave aquí es `run()`, una función suspendida que simula la carrera sin bloquear el hilo principal. En la interfaz de usuario, dentro de `RaceTrackerApp.kt`, se emplea `LaunchedEffect` para iniciar las corrutinas que ejecutan la función `run()` de cada participante cuando comienza la carrera. Para permitir que ambos corredores avancen al mismo tiempo, se utiliza `launch` dentro de `LaunchedEffect`, creando corrutinas separadas para cada uno. Además, se usa `coroutineScope` para coordinar la finalización de ambas carreras antes de actualizar el estado de la interfaz, como el cambio del texto del botón al finalizar la simulación.

En resumen, el codelab guía al usuario a través de la implementación de una función suspendida para simular el progreso de cada jugador, el uso de `LaunchedEffect` para iniciar y gestionar las corrutinas desde la interfaz de usuario, el empleo de `launch` para ejecutar las carreras de los jugadores de forma concurrente y el uso de `coroutineScope` para coordinar la finalización de ambas carreras antes de actualizar el estado de la interfaz. Además, se explica cómo manejar la cancelación de las corrutinas al interactuar con los botones de la aplicación.



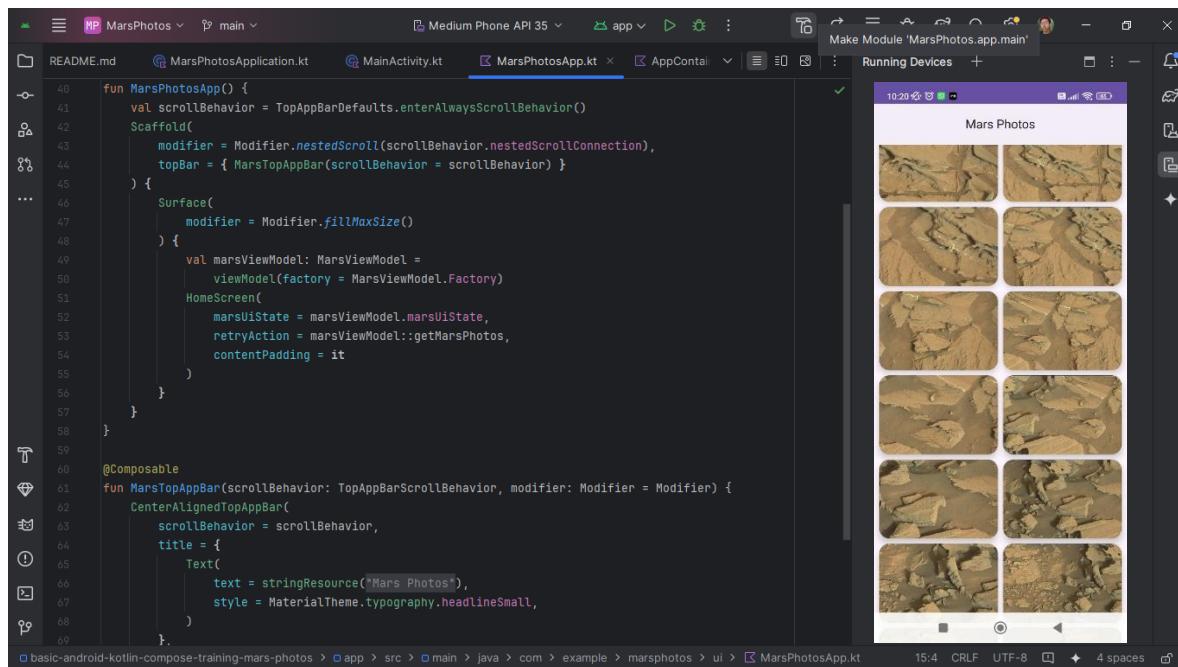
## Cómo obtener datos de Internet

En este codelab desarrollamos una aplicación llamada *Mars Photos*, que permite visualizar imágenes reales de la superficie de Marte, capturadas por rovers de la NASA. La app se conecta a un servicio web RESTful para obtener las fotos mediante una solicitud HTTP tipo GET y las presenta en una interfaz sencilla basada en Jetpack Compose.

Aprendimos a implementar una capa de datos que se comunica con un servidor backend a través de la biblioteca de terceros **Retrofit**, la cual facilita las conexiones HTTP y el consumo de servicios web. Para manejar las respuestas JSON y convertirlas en objetos de Kotlin utilizamos la biblioteca **kotlinx.serialization**, lo que nos permitió deserializar los datos obtenidos desde el servidor.

También trabajamos con un ViewModel que se encarga de realizar las llamadas de red y de actualizar el estado de la interfaz cuando se reciben nuevas fotos. Se implementaron prácticas recomendadas como la ejecución de operaciones en segundo plano y el manejo de excepciones de red para informar al usuario cuando hay problemas de conectividad.

Finalmente, otorgamos permisos a la app para conectarse a Internet y mostramos la cantidad de imágenes recibidas como confirmación del correcto funcionamiento de la capa de red. Este ejercicio nos ayudó a comprender cómo estructurar una app moderna de Android que interactúe con servicios web utilizando bibliotecas comunitarias bien mantenidas.



The screenshot shows the Android Studio interface with the MarsPhotos project open. The code editor displays the `MarsPhotosApp.kt` file, which contains the main application logic. The preview window on the right shows a 4x3 grid of Mars surface images. The status bar at the bottom indicates the device is a Medium Phone API 35, and the bottom right corner shows the current time as 10:20.

```
40 fun MarsPhotosApp() {
41     val scrollBehavior = TopAppBarDefaults.enterAlwaysScrollBehavior()
42     Scaffold(
43         modifier = Modifier.nestedScroll(scrollBehavior.nestedScrollConnection),
44         topBar = { MarsTopAppBar(scrollBehavior = scrollBehavior) }
45     ) {
46         Surface(
47             modifier = Modifier.fillMaxSize()
48         ) {
49             val marsViewModel: MarsViewModel =
50                 viewModel(factory = MarsViewModel.Factory)
51             HomeScreen(
52                 marsUiState = marsViewModel.marsUiState,
53                 retryAction = marsViewModel::getMarsPhotos,
54                 contentPadding = it
55             )
56         }
57     }
58 }
59
60 @Composable
61 fun MarsTopAppBar(scrollBehavior: TopAppBarScrollBehavior, modifier: Modifier = Modifier) {
62     CenterAlignedTopAppBar(
63         scrollBehavior = scrollBehavior,
64         title = {
65             Text(
66                 text = stringResource("Mars Photos"),
67                 style = MaterialTheme.typography.headlineSmall,
68             )
69         }
70     )
71 }
```

## Ruta de Aprendizaje 2 (Cómo cargar y mostrar imágenes de Internet)

### Cómo agregar el repositorio y la inserción manual de dependencias

En esta etapa del proyecto, se implementaron mejoras en la arquitectura del código enfocadas exclusivamente al **backend**, sin modificar la interfaz de usuario.

Los principales cambios fueron:

- **Separación de responsabilidades** mediante la introducción del **patrón de repositorio**, que abstrae el acceso a los datos y desacopla la lógica de red del ViewModel.
- **Uso de interfaces** para facilitar el intercambio de implementaciones (por ejemplo, para pruebas).

- **Aplicación del principio de inyección de dependencias (DI)**, lo cual mejora la escalabilidad y facilita las pruebas unitarias.
- **Refactorización del ViewModel** para que ya no se encargue directamente de obtener los datos, sino que reciba una fuente de datos externa (repositorio).

Estos cambios mejoran la mantenibilidad, la escalabilidad y la testabilidad del código, siguiendo buenas prácticas de arquitectura moderna en Android.

### **Cómo cargar y mostrar imágenes de Internet**

En esta actividad se implementó la carga y visualización de imágenes desde una URL en una aplicación Android utilizando **Jetpack Compose**. Se integró la biblioteca **Coil**, la cual permite descargar, decodificar, almacenar en caché y mostrar imágenes de forma eficiente en segundo plano, sin afectar el rendimiento de la interfaz.

### **Principales tareas realizadas**

- Se agregó la dependencia de **Coil Compose** en el archivo build.gradle.kts.
- Se utilizó el componible **Asynclmage** para mostrar imágenes de Marte descargadas desde un servicio web.
- Se incorporaron imágenes de marcador de posición (loading) y error para mejorar la experiencia de usuario.
- Se implementó el componente **LazyVerticalGrid** con GridCells.Adaptive(150.dp) para mostrar una cuadrícula de imágenes que se adapta dinámicamente al tamaño de pantalla.
- Se incluyeron **claves de elementos (keys)** para preservar el estado y la posición de desplazamiento al reordenar elementos o cambiar la orientación del dispositivo.
- Se añadió una acción de "**Reintentar**" mediante un botón que vuelve a realizar la solicitud de carga si falla.
- Se actualizó el ViewModel y su prueba unitaria para trabajar con una lista de fotos en lugar de una sola imagen.

La aplicación ahora es capaz de mostrar una cuadrícula de imágenes de Marte descargadas desde la red, de forma responsive, eficiente y resiliente a errores, cumpliendo con buenas prácticas de UI y rendimiento en Android.

## Práctica: Compila App de Anfibios

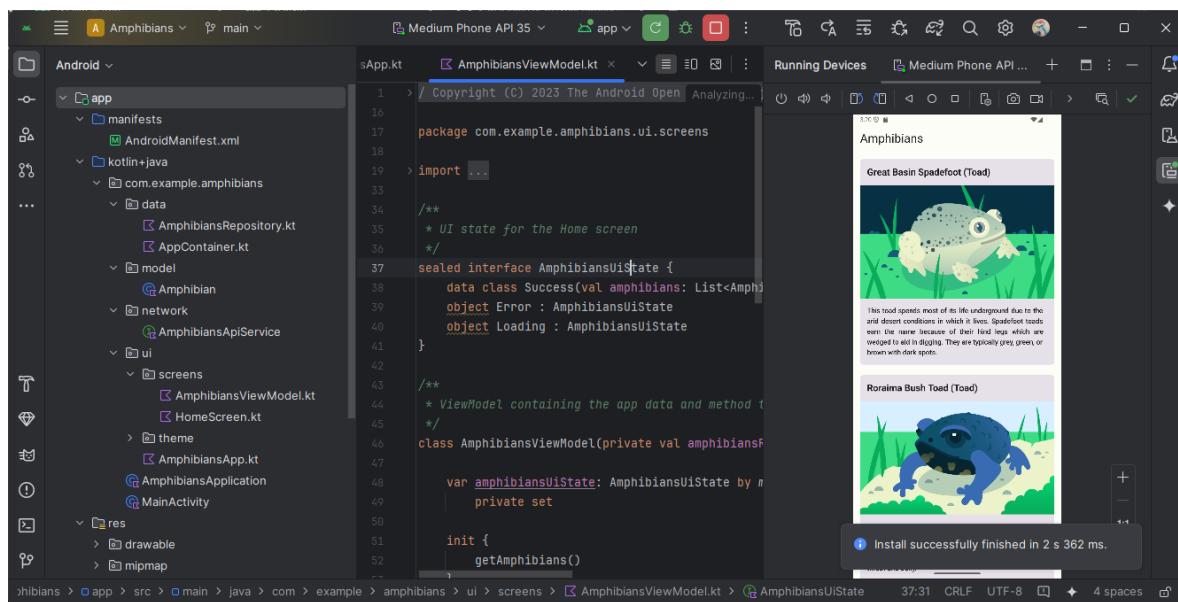
Aquí se recupera y se muestra información sobre anfibios desde una API web. La aplicación muestra una lista desplazable de anfibios, incluyendo su nombre, tipo, descripción e imagen obtenida de una URL.

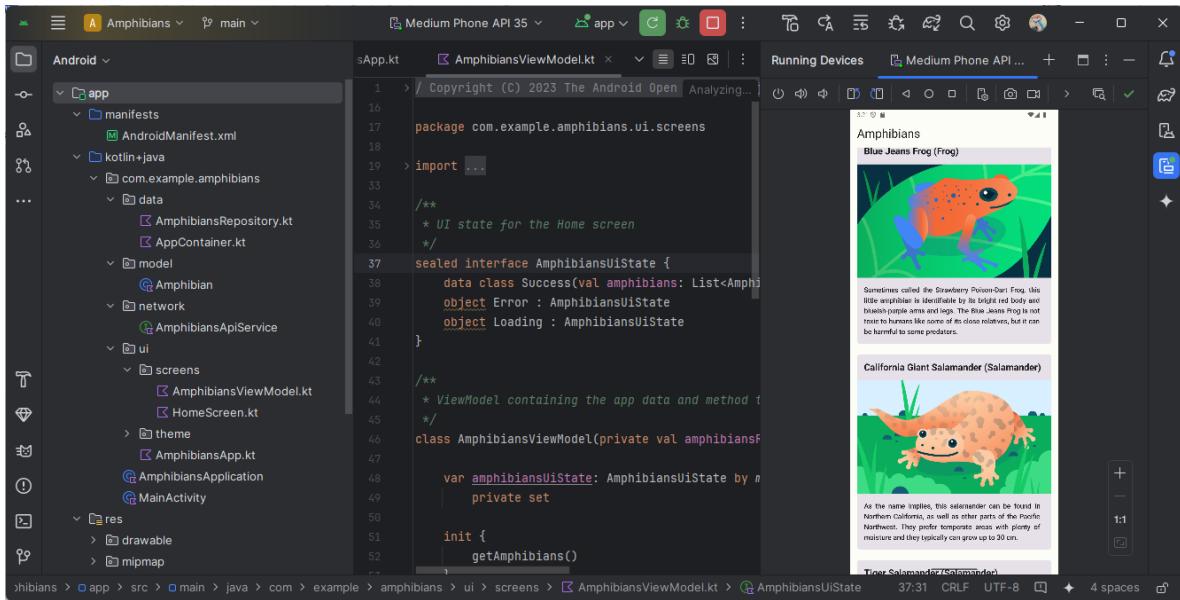
El proceso de desarrollo se enfoca en la creación de una capa de datos responsable de la comunicación con la API, la cual incluye una clase de datos para la información de los anfibios, un repositorio para gestionar los datos y una fuente de datos para realizar las peticiones de red utilizando Retrofit y para analizar la respuesta JSON con kotlinx.serialization.

La estructura del paquete com.example.amphibians está organizada en capas, donde la actividad principal MainActivity.kt, configura la interfaz utilizando funciones composable. El archivo Amphibian.kt define el modelo de datos que representa a cada anfibio. En la parte de red, Amphibian ApiService.kt implementa Retrofit para realizar solicitudes a la API. Toda la lógica de negocio y el manejo de estado están centralizados en el AmphibianViewModel.kt, mientras que AmphibianUiState.kt define los distintos estados de la interfaz (como cargando, error o datos disponibles).

Por otra parte, AmphibianCard.kt define el componente visual que muestra cada tarjeta de anfibio con su imagen e información, y AmphibianListScreen.kt organiza cómo se presenta la lista completa.

En resumen, el objetivo del ejercicio es construir una aplicación que consume datos de una API para mostrar una lista de anfibios con sus detalles e imágenes, aplicando patrones de arquitectura como la separación de la capa de datos y la inyección de dependencias para un desarrollo más organizado y mantenable.





**Nota:** Los códigos con comentarios se encuentran subidos en [github](#).

### Proyecto: Crea una app de Bookshelf

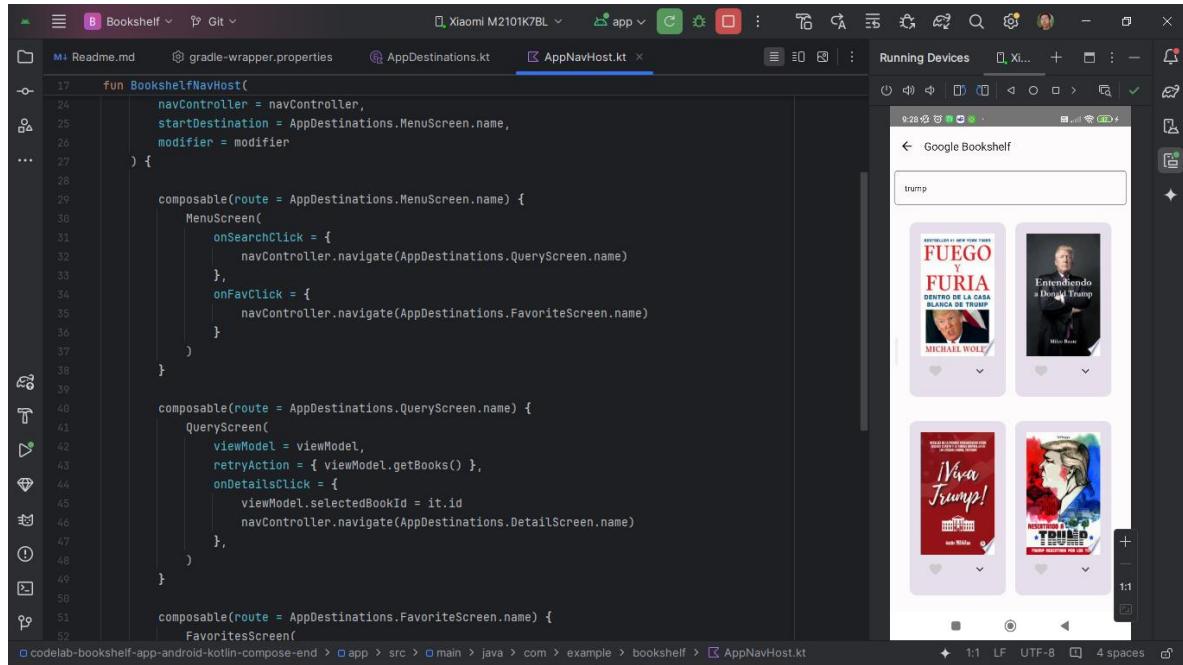
Durante el desarrollo de la unidad 5, se construyó una aplicación Android llamada **Bookshelf**, cuyo propósito es practicar e integrar conceptos clave como el uso de **corrutinas**, **Retrofit**, **Gson**, **inyección de dependencias** y **Coil** para la descarga asíncrona de imágenes.

La app permite realizar búsquedas de libros a través de la **API de Google Books** y mostrar los resultados en una cuadrícula visualmente atractiva con la imagen de portada y el título de cada libro.

### Componentes principales

- Se creó una interfaz de Retrofit que realiza peticiones HTTP a la API pública de Google Books.
- Se configuró el uso de **Gson** para analizar las respuestas en formato JSON y extraer los datos relevantes, como el id, el título y la URL de la imagen (thumbnail).
- Se procesó correctamente la URL de imagen reemplazando http por https para asegurar que las imágenes se cargaran correctamente.
- Se implementó un **repositorio** que actúa como capa intermedia entre la UI y la red.
- Esta capa abstracta permitió aplicar buenas prácticas de arquitectura, facilitando la **prueba y el mantenimiento** del código.
- Se inyectó el repositorio mediante **inyección de dependencias**, lo que permite intercambiar entre un servicio real y uno simulado (falso/mock).
- Se diseñó un **servicio falso (fake)** para simular respuestas de la API.

- Se realizaron pruebas unitarias sobre el repositorio utilizando el servicio falso, verificando la correcta manipulación de los datos.
- Se usó **Jetpack Compose** para construir la interfaz.
- Los resultados se mostraron en una **cuadrícula vertical** con desplazamiento, en la que cada celda contenía una imagen de la portada del libro y su título.
- Las imágenes se descargaron de manera asíncrona usando el componible `AsyncImage` de la biblioteca **Coil**, lo cual optimizó el rendimiento y la experiencia del usuario.



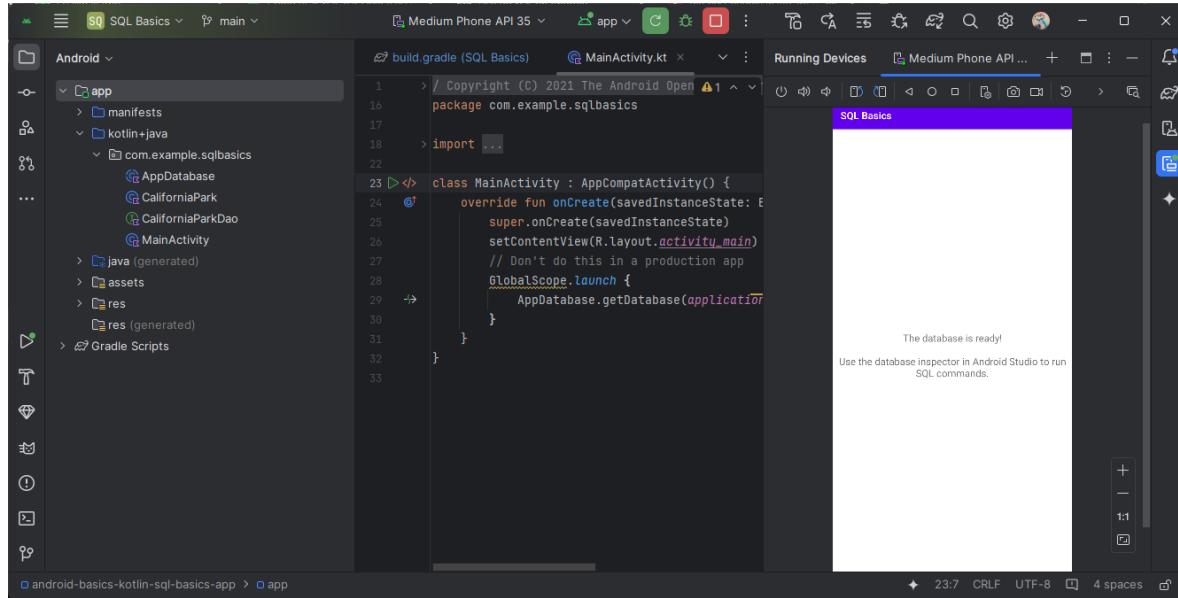
The screenshot shows the Android Studio interface with the following details:

- Left Panel (Code View):** The file `AppNavHost.kt` is open, displaying Kotlin code for a navigation host. The code uses `composable` blocks to define routes for different screens: `MenuScreen`, `QueryScreen`, and `FavoritesScreen`. Each screen has associated click listeners for search and favorite actions.
- Top Bar:** Shows the project name "Bookshelf", the branch "Git", the device "Xiaomi M2101K7BL", and the tab "app".
- Right Panel (Running Devices):** Shows a running instance of the "Google Bookshelf" app on the selected device. The app's search bar contains the text "trump". Below it, several book covers are displayed in a grid, including titles like "FUEGO Y FURIA", "Entiendo a Donald Trump", "Viva Trump!", and "Reservados a Trump".
- Bottom Status Bar:** Shows the zoom level (1:1), file type (LF), encoding (UTF-8), and code style settings (4 spaces).

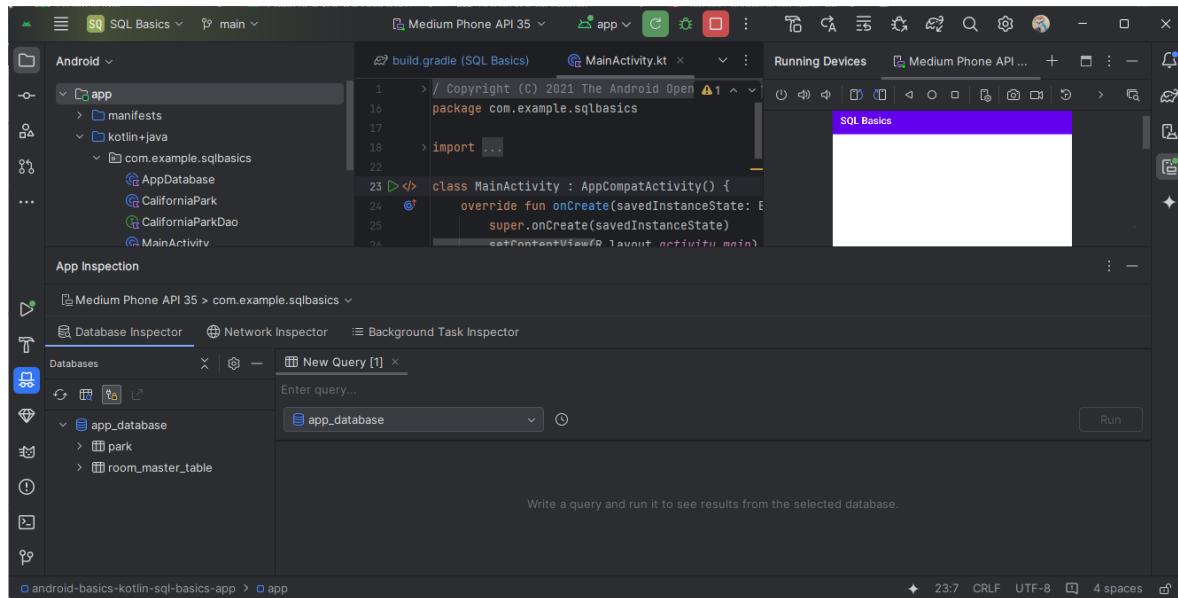
## Unidad 6: Persistencia de Datos

### Ruta de Aprendizaje 1 (Introducción a SQL)

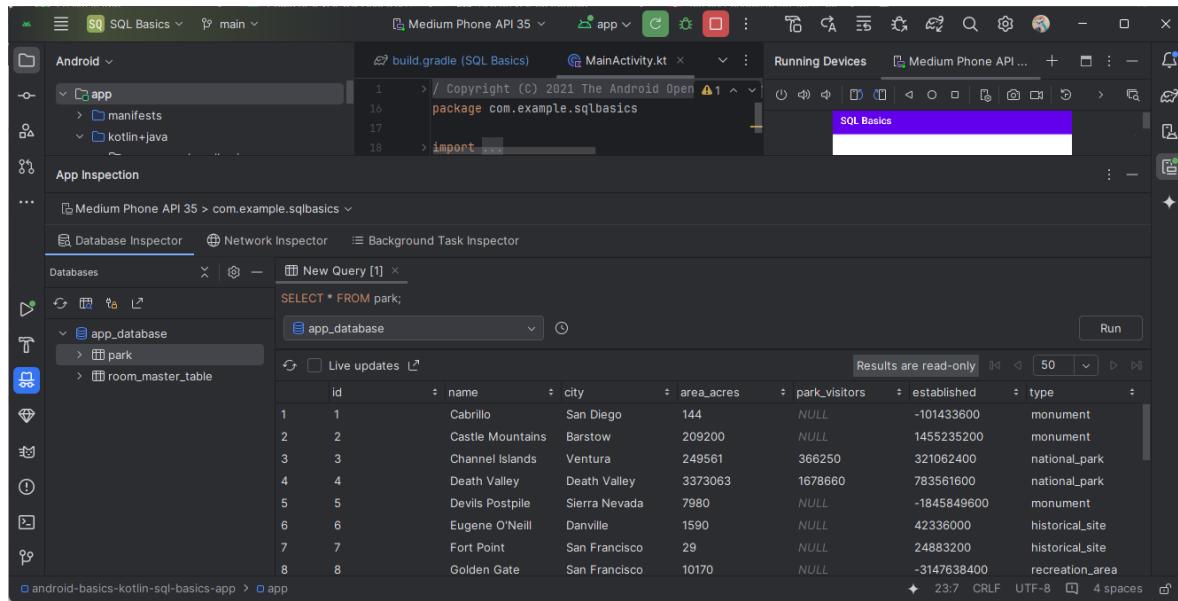
Descargamos el proyecto inicial y ejecutamos en Android Studio:



Posteriormente nos dirigimos a App Inspection con la pestaña Database Inspector, donde el proyecto actual está configurado para usar una base de datos diferente a la del ejemplo del curso. En lugar de una app de correos, está usando una app de parques nacionales (tabla park en vez de tabla email) porque el repositorio fue actualizado con otro ejemplo. Sin embargo, el funcionamiento sigue siendo el mismo.

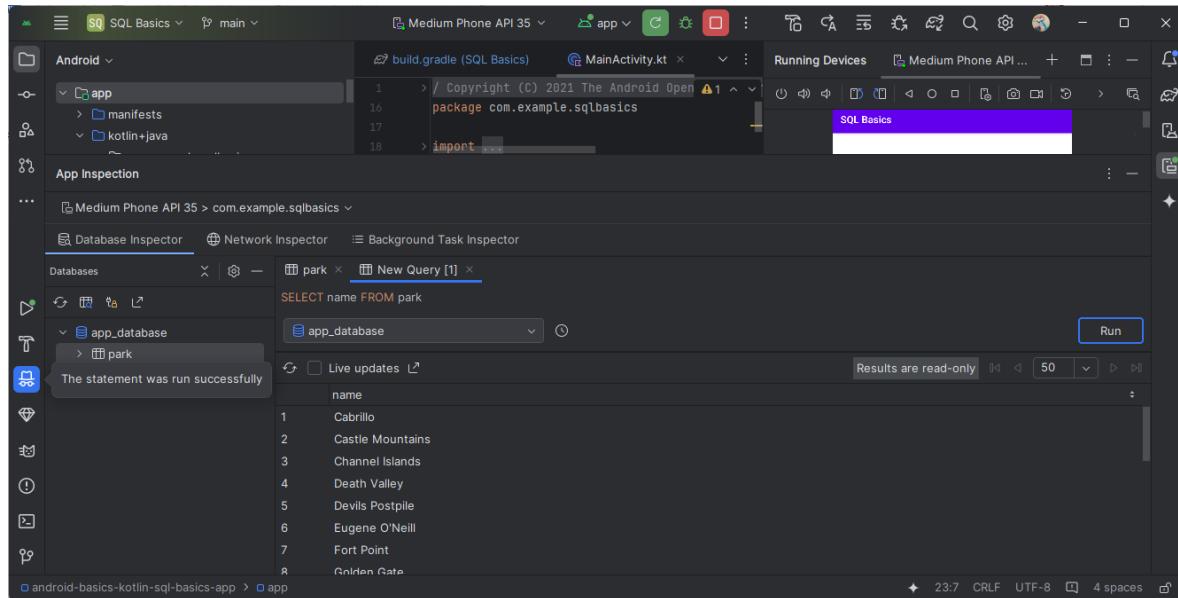


Hacemos uso de la instrucción SELECT de SQL y vemos que se muestra toda la tabla de park (en este caso):



	id	name	city	area_acres	park_visitors	established	type
1	1	Cabrillo	San Diego	144	NULL	-101433600	monument
2	2	Castle Mountains	Barstow	209200	NULL	1455235200	monument
3	3	Channel Islands	Ventura	249561	366250	321062400	national_park
4	4	Death Valley	Death Valley	3373063	1678660	783561600	national_park
5	5	Devils Postpile	Sierra Nevada	7980	NULL	-1845849600	monument
6	6	Eugene O'Neill	Darville	1590	NULL	42336000	historical_site
7	7	Fort Point	San Francisco	29	NULL	24883200	historical_site
8	8	Golden Gate	San Francisco	10170	NULL	-3147638400	recreation_area

Ahora seleccionamos solo una columna. En el curso es SELECT subject FROM email; en nuestro caso es SELECT name FROM park



	name
1	Cabrillo
2	Castle Mountains
3	Channel Islands
4	Death Valley
5	Devils Postpile
6	Eugene O'Neill
7	Fort Point
8	Golden Gate

Y para seleccionar varias columnas en el ejemplo es SELECT subject, sender FROM email; y en el nuestro es SELECT name, city FROM park

The screenshot shows the Android Studio interface with the Database Inspector open. The left sidebar shows the project structure with an 'app' module selected. In the Database Inspector, the 'park' table is selected. A query is being run:

```
SELECT name, city FROM park
```

The results table shows the following data:

	name	city
1	Cabrillo	San Diego
2	Castle Mountains	Barstow
3	Channel Islands	Ventura
4	Death Valley	Death Valley
5	Devils Postpile	Sierra Nevada
6	Eugene O'Neill	Danville
7	Fort Point	San Francisco
8	Golden Gate	San Francisco

Para contar la cantidad de parques en la base de datos usamos `SELECT COUNT(*)` FROM park:

The screenshot shows the Android Studio interface with the Database Inspector open. The left sidebar shows the project structure with an 'app' module selected. In the Database Inspector, the 'park' table is selected. A query is being run:

```
SELECT COUNT(*) FROM park
```

The results table shows the following data:

	COUNT(*)
1	23

Para obtener el área más grande de los parques (usando MAX) en caso de que se quiera saber cuál es el parque con el área más grande, se usa MAX() sobre la columna area\_acres:

```
build.gradle (SQL Basics) MainActivity.kt
1 > / Copyright (C) 2021 The Android Open
2 package com.example.sqlbasics
3
4 import ...
```

The statement was run successfully

	MAX(area_acres)
1	3373063

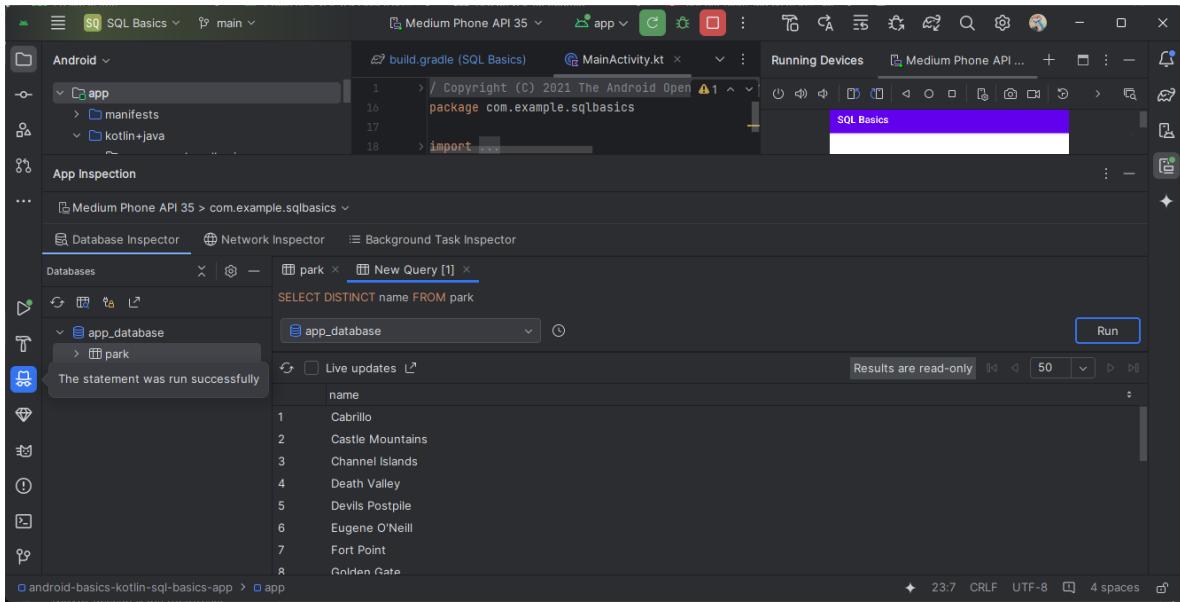
Ahora para filtrar los resultados duplicados con DISTINCT en el ejemplo original, la consulta SELECT sender FROM email; devuelve todos los valores de la columna sender (aunque haya duplicados). Para adaptarlo a la tabla park se selecciona la columna name.

```
build.gradle (SQL Basics) MainActivity.kt
1 > / Copyright (C) 2021 The Android Open
2 package com.example.sqlbasics
3
4 import ...
```

The statement was run successfully

	name
1	Cabrillo
2	Castle Mountains
3	Channel Islands
4	Death Valley
5	Devils Postpile
6	Eugene O'Neill
7	Fort Point
8	Golden Gate

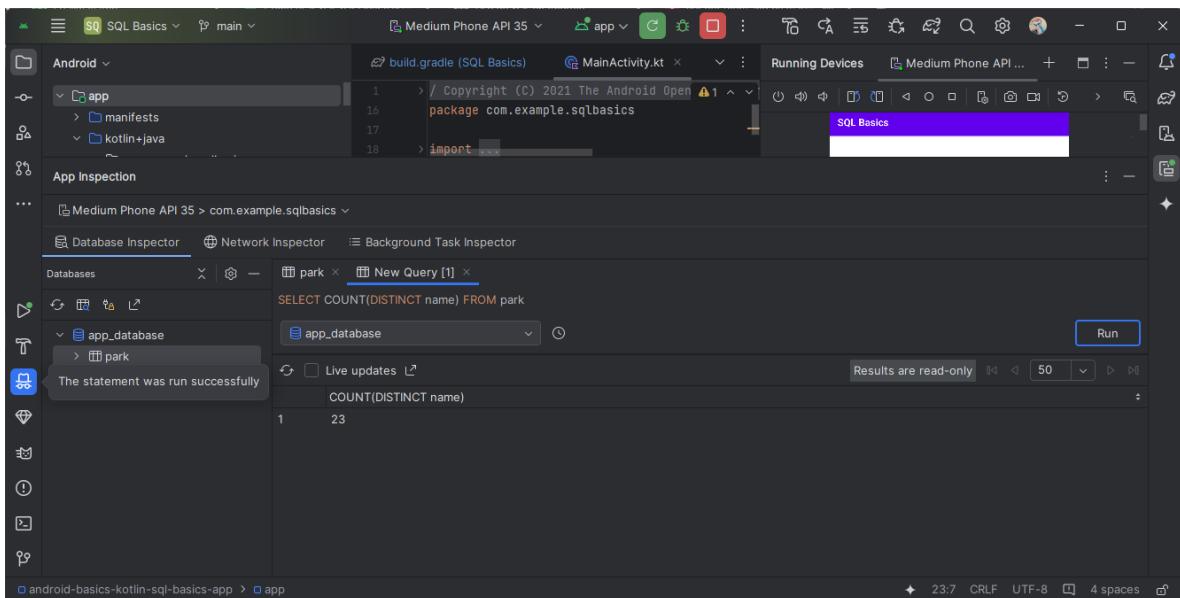
Ahora, en el ejemplo original, SELECT DISTINCT sender FROM email; selecciona los valores únicos de la columna sender. Como nosotros contamos con la tabla park y queremos ver los nombres de parques únicos (sin duplicados) se ejecuta lo siguiente:



The screenshot shows the Android Studio interface with the Database Inspector tool open. The left sidebar shows the project structure with an 'app' module selected. In the Database Inspector, the 'park' table is selected under the 'app\_database' database. A query is being run: 'SELECT DISTINCT name FROM park'. The results show 23 distinct names: Cabrillo, Castle Mountains, Channel Islands, Death Valley, Devils Postpile, Eugene O'Neill, Fort Point, and Golden Gate.

name
Cabrillo
Castle Mountains
Channel Islands
Death Valley
Devils Postpile
Eugene O'Neill
Fort Point
Golden Gate

En el ejemplo original, `SELECT COUNT(DISTINCT sender) FROM email;` cuenta cuántos remitentes únicos (`sender`) hay en la tabla `email`. Para la tabla `park` contamos cuántos nombres únicos de parques hay:



The screenshot shows the Android Studio interface with the Database Inspector tool open. The left sidebar shows the project structure with an 'app' module selected. In the Database Inspector, the 'park' table is selected under the 'app\_database' database. A query is being run: 'SELECT COUNT(DISTINCT name) FROM park'. The results show a single row with the value 23.

COUNT(DISTINCT name)
23

Un ejemplo básico con `WHERE` para la tabla `park`, si queremos filtrar los parques que están en una ciudad específica (por ejemplo, "San Diego"), la consulta se vería así: `SELECT * FROM park WHERE city = 'San Diego'`

The screenshot shows the Android Studio interface with the Database Inspector tool open. The query being run is:

```
WHERE city = 'San Diego'
```

The results show one row from the 'park' table:

	id	name	city	area_acres	park_visitors	established	type
	1	Cabrillo	San Diego	144	NULL	-101433600	monument

En cambio, para filtrar con varias condiciones usando AND en el ejemplo original se mostró cómo filtrar los correos electrónicos que estén en la carpeta "inbox" y que no hayan sido leídos, utilizando AND. Para la tabla park filtramos parques que estén en una ciudad específica y que también tengan más de 1000 visitantes:

The screenshot shows the Android Studio interface with the Database Inspector tool open. The query being run is:

```
SELECT * FROM park  
WHERE city = 'Ventura' AND park_visitors > 1000
```

The results show one row from the 'park' table:

	id	name	city	area_acres	park_visitors	established	type
	1	Channel Islands	Ventura	249561	366250	321062400	national_park

Para filtrar con OR filtramos los parques que están en "Palms" o en "San Francisco":

The screenshot shows the Android Studio interface with the 'SQL Basics' project open. In the Database Inspector, a query is run against the 'park' table:

```
SELECT * FROM park  
WHERE city = 'Palms' OR city = 'San Francisco'
```

The results show four rows of data from the 'park' table:

	id	name	city	area_acres	park_visitors	established	type
1	7	Fort Point	San Francisco	29	NULL	24883200	historical_site
2	8	Golden Gate	San Francisco	10170	NULL	-3147638400	recreation_area
3	10	Joshua Tree	Palms	790636	2942381	783561600	national_park
4	18	San Francisco Mariti	San Francisco	50	4223542	583372800	national_park

Si queremos filtrar con NOT para seleccionar los parques que no están en "San Francisco":

The screenshot shows the Android Studio interface with the 'SQL Basics' project open. In the Database Inspector, a query is run against the 'park' table:

```
SELECT * FROM park  
WHERE NOT city = 'San Francisco'
```

The results show seven rows of data from the 'park' table, excluding the two rows for San Francisco:

	id	name	city	area_acres	park_visitors	established	type
1	1	Cabrillo	San Diego	144	NULL	-101433600	monument
2	2	Castle Mountains	Barstow	209200	NULL	1455235200	monument
3	3	Channel Islands	Ventura	249561	366250	321062400	national_park
4	4	Death Valley	Death Valley	3373063	1678660	783561600	national_park
5	5	Devils Postpile	Sierra Nevada	7980	NULL	-1845849600	monument
6	6	Eugene O'Neill	Danville	1590	NULL	42336000	historical_site
7	9	Inho Muir	Martinez	3450	NULL	-101433600	historical_site

Para buscar texto con LIKE en este caso buscamos parques cuyo nombre contiene la palabra "Fort" (en cualquier parte del nombre):

The screenshot shows the Android Studio interface with the Database Inspector open. A query is being run against the 'park' table in the 'app\_database'. The results show one row with id 7, name 'Fort Point', city 'San Francisco', area\_acres 29, park\_visitors NULL, established 24883200, and type 'historical\_site'.

```
SELECT * FROM park  
WHERE name LIKE '%Fort%';
```

	id	name	city	area_acres	park_visitors	established	type
	7	Fort Point	San Francisco	29	NULL	24883200	historical_site

Para contar cuántos parques tienen la palabra "fool" en el nombre:

The screenshot shows the Android Studio interface with the Database Inspector open. A query is being run against the 'park' table in the 'app\_database'. The results show a single row with COUNT(\*) as 0.

```
SELECT COUNT(*) FROM park  
WHERE name LIKE '%fool%';
```

COUNT(*)
0

Mostrar todos los datos de los parques cuyo nombre termina con "fool":

```
SELECT * FROM park  
WHERE name LIKE '%fool'
```

The statement was run successfully

	id	name	city	area_acres	park_visitors	established	type
Table is empty							

Mostrar los nombres distintos de parques que comienzan con la letra "r":

```
SELECT DISTINCT name FROM park  
WHERE name LIKE 'r%'
```

The statement was run successfully

name
Redwood
Rosie the Riveter WWII Home Front

Para agrupar los resultados con GROUP BY, por ejemplo: ¿cuántos parques hay por tipo (como "National", "Urban", etc.)?

The screenshot shows the Android Studio interface with the Database Inspector open. A query is being run against the 'park' table:

```
SELECT type, COUNT(*) AS total_parks FROM park GROUP BY type
```

The results show the count of parks by type:

type	total_parks
historical_site	5
monument	6
national_park	9
recreation_area	3

Para ordenar los resultados con ORDER BY y mostrar todos los parques ordenados por número de visitantes, del más visitado al menos visitado:

The screenshot shows the Android Studio interface with the Database Inspector open. A query is being run against the 'park' table:

```
SELECT * FROM park ORDER BY park_visitors DESC
```

The results show the parks ordered by visitors in descending order:

id	name	city	area_acres	park_visitors	established	type
18	San Francisco Mariti.	San Francisco	50	4223542	583372800	national_park
23	Yosemite	Yosemite	748436	4161087	-2500934400	national_park
10	Joshua Tree	Palms	790636	2942381	783561600	national_park
4	Death Valley	Death Valley	3373063	1678660	783561600	national_park
11	Lassen Volcanic	Mineral	106452	499435	-1685059200	national_park
16	Redwood	Crescent City	138999	482536	-39398400	national_park
20	Seniora & Kinn	Car Fresno	461011	415077	-2500934400	national_park

Para mostrar todos los parques ordenados por superficie (area\_acres), del más pequeño al más grande:

The screenshot shows the Android Studio interface with the Database Inspector open. A query is running against the 'park' table:

```
SELECT * FROM park  
ORDER BY area_acres ASC
```

The results show 21 rows of park data. The columns are: id, name, city, area\_acres, park\_visitors, established, and type. The data includes various national parks like San Francisco Mariti, Yosemite, and Death Valley, along with historical sites like Muir Woods and Rosie the Riveter W' Richmond.

	id	name	city	area_acres	park_visitors	established	type
1	7	Fort Point	San Francisco	29	NULL	24883200	historical_site
2	18	San Francisco Mariti	San Francisco	50	4223542	583372800	national_park
3	1	Cabrillo	San Diego	144	NULL	-101433600	monument
4	17	Rosie the Riveter W' Richmond		145	NULL	980899200	historical_site
5	14	Muir Woods	Mill Valley	554	NULL	-1955923200	monument
6	13	Manzanar	Independence	814	NULL	207532800	historical_site
7	21	Title Lake	Title Lake	1391	NULL	1140134400	monument

Para limitar resultados con LIMIT y mostrar los 5 parques más visitados:

The screenshot shows the Android Studio interface with the Database Inspector open. A query is running against the 'park' table:

```
SELECT * FROM park  
ORDER BY park_visitors DESC  
LIMIT 5
```

The results show 5 rows of park data, ordered by visitors from highest to lowest. The columns are: id, name, city, area\_acres, park\_visitors, established, and type. The data includes San Francisco Mariti, Yosemite, Joshua Tree, Death Valley, and Lassen Volcanic.

	id	name	city	area_acres	park_visitors	established	type
1	18	San Francisco Mariti	San Francisco	50	4223542	583372800	national_park
2	23	Yosemite	Yosemite	748436	4161087	-2500934400	national_park
3	10	Joshua Tree	Palms	790636	2942381	783561600	national_park
4	4	Death Valley	Death Valley	3373063	1678660	783561600	national_park
5	11	Lassen Volcanic	Mineral	106452	499435	-1685059200	national_park

Para mostrar los parques del 6º al 10º más visitados:

```
SELECT * FROM park
ORDER BY park_visitors DESC
LIMIT 5 OFFSET 5
```

	id	name	city	area_acres	park_visitors	established	type
1	16	Redwood	Crescent City	138999	482536	-39398400	national_park
2	20	Sequoia & Kings Can	Fresno	461901	415077	-250093440	national_park
3	3	Channel Islands	Ventura	249561	366250	321062400	national_park
4	15	Pinnacles	Placines	26606	222152	1357776000	national_park
5	1	Cabrillo	San Diego	144	NULL	-101433600	monument

Para insertar datos:

```
INSERT INTO park
VALUES (
    NULL, 'Bosque Encantado', 'Tepoztlán', 120, 85000, 1714785600000, 'natural_park'
);
```

The statement was run successfully

The screenshot shows the Android Studio interface with the Database Inspector tool open. A query named 'park' is running, displaying the following data:

	id	name	city	area_acres	park_visitors	established	type
14	14	Muir Woods	Mill Valley	554	NULL	-1955923200	monument
15	15	Pinnacles	Placines	26606	222152	1357776000	national_park
16	16	Redwood	Crescent City	138999	482536	-39398400	national_park
17	17	Rosie the Riveter W' Richmond		145	NULL	980899200	historical_site
18	18	San Francisco Mariti	San Francisco	50	4223542	583372800	national_park
19	19	Santa Monica Moun	Thousand Oaks	156671	NULL	279504000	recreation_area
20	20	Sequoia & Kings Car	Fresno	461901	415077	-250093440	national_park
21	21	Tule Lake	Tule Lake	1391	NULL	1140134400	monument
22	22	Whiskeytown	Whiskeytown	203587	NULL	-130896000	recreation_area
23	23	Yosemite	Yosemite	748436	4161087	-250093440	national_park
24	24	Bosque Encantado	Tepoztlán	120	85000	1714785600000	natural_park

El resultado se inserta en la base de datos con un id de 24:

The screenshot shows the Android Studio interface with the Database Inspector tool open. A query named 'park' is running, displaying the following data:

	id	name	city	area_acres	park_visitors	established	type
1	24	Bosque Encantado	Tepoztlán	120	85000	1714785600000	natural_park

Actualizamos el registro con un UPDATE:

The screenshot shows the Android Studio interface with the Database Inspector tool open. In the Database Inspector, under the 'app\_database' section, there is a table named 'park'. A query is being run against this table:

```
UPDATE park SET park_visitors = 90000 WHERE id = 24
```

The result of the query is displayed below the query editor:

The statement was run successfully

The screenshot shows the Android Studio interface with the Database Inspector tool open. In the Database Inspector, under the 'app\_database' section, there is a table named 'park'. The table now contains the following data:

	id	name	city	area_acres	park_visitors	established	type
14	14	Muir Woods	Mill Valley	554	NULL	-1955923200	monument
15	15	Pinnacles	Placines	26606	222152	1357776000	national_park
16	16	Redwood	Crescent City	138999	482536	-39398400	national_park
17	17	Rosie the Riveter W	Richmond	145	NULL	980899200	historical_site
18	18	San Francisco Maril	San Francisco	50	4223542	583372800	national_park
19	19	Santa Monica Moun	Thousand Oaks	156671	NULL	279504000	recreation_area
20	20	Sequoia & Kings Car	Fresno	461901	415077	-2500934400	national_park
21	21	Tule Lake	Tule Lake	1391	NULL	1140134400	monument
22	22	Whiskeytown	Whiskeytown	203587	NULL	-130896000	recreation_area
23	23	Yosemite	Yosemite	748436	4161087	-2500934400	national_park
24	24	Bosque Encantado	Tepoztlán	120	90000	1714785600000	natural_park

Por otra parte, si queremos eliminar el registro:

The screenshot shows the Android Studio interface with the project 'SQL Basics' open. In the bottom right corner of the code editor, there is a purple status bar with the text 'SQL Basics'. The code editor displays the following Java code:

```
Copyright (C) 2021 The Android Open Source Project
package com.example.sqlbasics
import ...
```

In the Database Inspector tab, under the 'app\_database' database, there is a table named 'park'. A query is being run against it:

```
DELETE FROM park WHERE id = 24
```

The results of the query are shown in the results pane:

	id	name	city	area_acres	park_visitors	established	type
The statement was run successfully.							

The message 'The statement was run successfully' is displayed below the results.

Verificamos que fue eliminado:

The screenshot shows the Android Studio interface with the project 'SQL Basics' open. The Database Inspector tab is active, showing the 'app\_database' database and the 'park' table. A new query is being run:

```
SELECT * FROM park WHERE id = 24
```

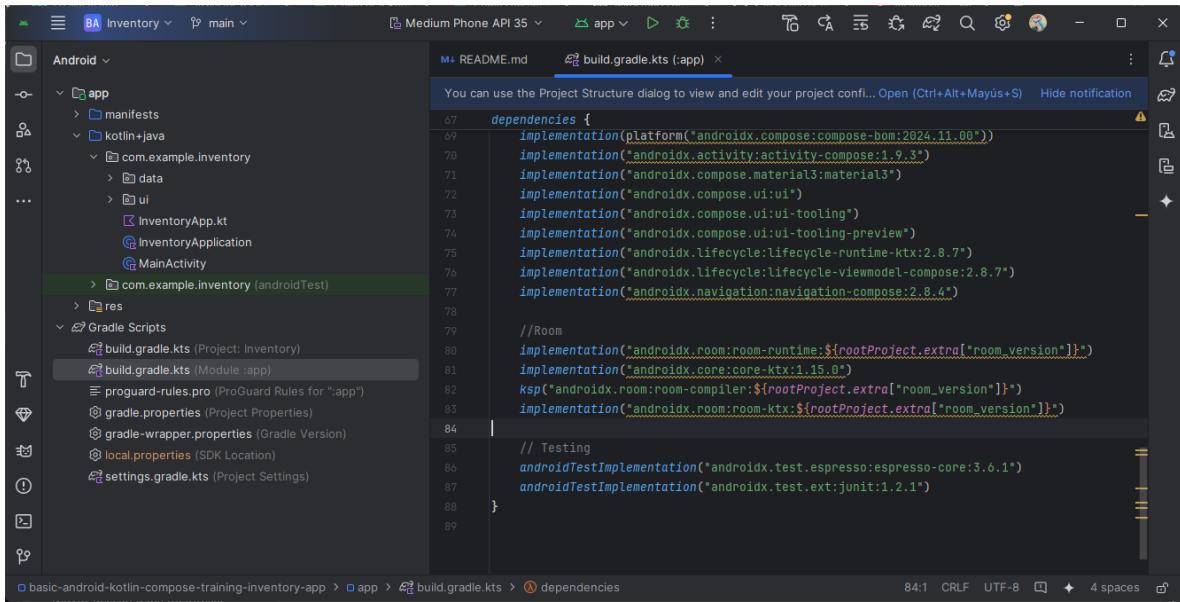
The results of the query are shown in the results pane:

	id	name	city	area_acres	park_visitors	established	type
Live updates							
Results are read-only	50						
Table is empty							

The message 'Table is empty' is displayed below the results.

## Ruta de Aprendizaje 2 (Cómo usar Room para lograr la persistencia de datos)

Lo primero que se hace en el desarrollo de la actividad es agregar las dependencias de Room en nuestro proyecto:



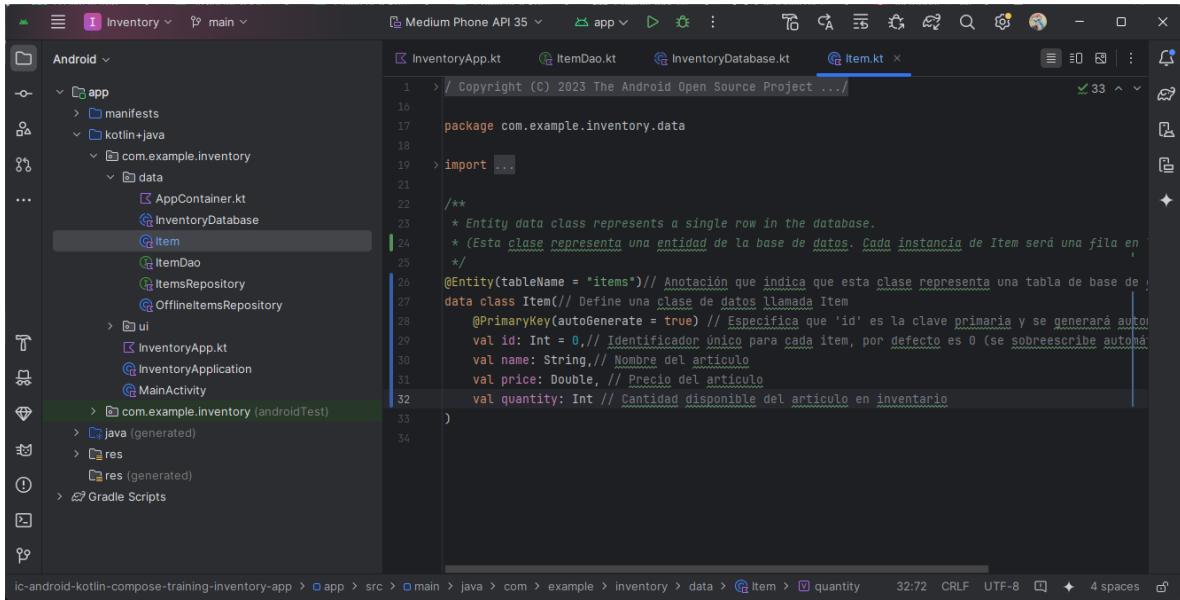
The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right. The code editor displays the `build.gradle.kts` file for the `app` module. The file contains dependencies for Room and Room-Ktx, as well as androidTestImplementation for Espresso and JUnit.

```
dependencies {
    implementation("androidx.compose:compose-bom:2024.11.00")
    implementation("androidx.activity:activity-compose:1.9.3")
    implementation("androidx.compose.material3:material3")
    implementation("androidx.compose.ui:ui")
    implementation("androidx.compose.ui:ui-tooling")
    implementation("androidx.compose.ui:ui-tooling-preview")
    implementation("androidx.lifecycle:lifecycle-runtime-ktx:2.8.7")
    implementation("androidx.lifecycle:lifecycle-viewmodel-compose:2.8.7")
    implementation("androidx.navigation:navigation-compose:2.8.4")

    // Room
    implementation("androidx.room:room-runtime:${rootProject.extra["room_version"]}")
    implementation("androidx.core:core-ktx:1.15.0")
    ksp("androidx.room:room-compiler:${rootProject.extra["room_version"]}")
    implementation("androidx.room:room-ktx:${rootProject.extra["room_version"]}")

    // Testing
    androidTestImplementation("androidx.test.espresso:espresso-core:3.6.1")
    androidTestImplementation("androidx.test.ext:junit:1.2.1")
}
```

Seguimos los pasos necesarios como la definición de la clase “Item” como entidad que representa los datos que se almacenarán en la base de datos. La interfaz “ItemDao”, que actúa como el Data Access Object (DAO) y define métodos para insertar, actualizar, eliminar y obtener elementos de la base de datos. En cuanto a “InventoryDatabase” esta se configura mediante la clase abstracta “RoomDatabase”, la cual proporciona acceso al DAO:



The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right. The code editor displays the `Item.kt` file, which is part of the `com.example.inventory.data` package. The file defines the `Item` class with annotations for Room, including `@Entity` and `@PrimaryKey`.

```
package com.example.inventory.data

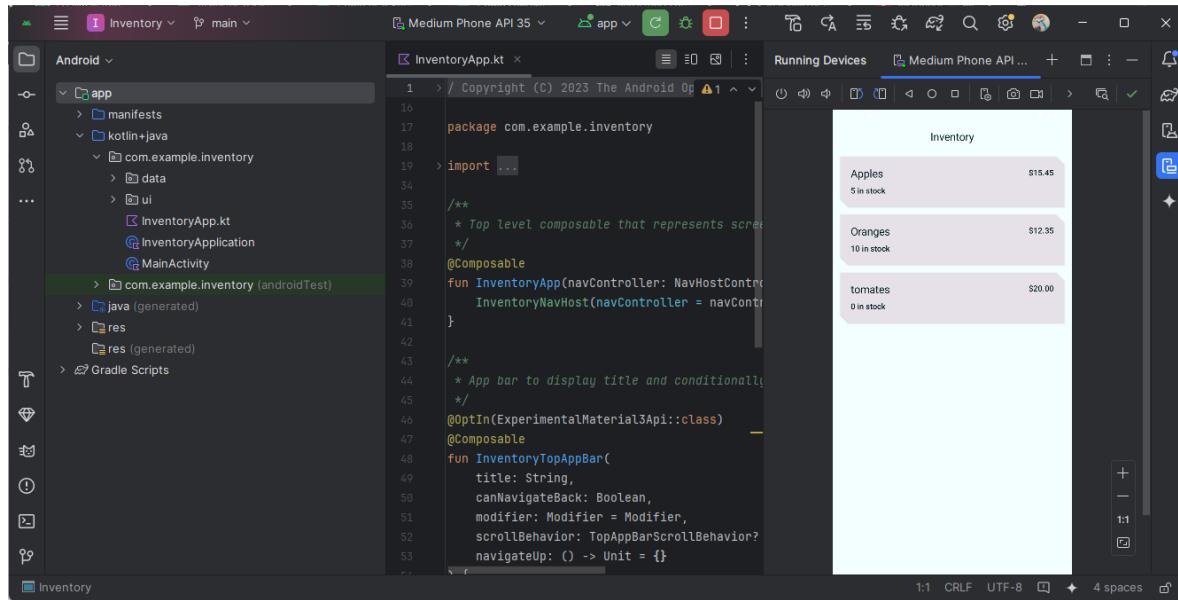
import ...

/**
 * Entity data class represents a single row in the database.
 * (Esta clase representa una entidad de la base de datos. Cada instancia de Item será una fila en la tabla items.)
 */
@Entity(tableName = "items")// Anotación que indica que esta clase representa una tabla de base de datos
data class Item// Define una clase de datos llamada Item
    @PrimaryKey(autoGenerate = true) // Especifica que 'id' es la clave primaria y se generará automáticamente
    val id: Int = 0, // Identificador único para cada ítem, por defecto es 0 (se sobreescribe automáticamente)
    val name: String, // Nombre del artículo
    val price: Double, // Precio del artículo
    val quantity: Int // Cantidad disponible del artículo en inventario
```

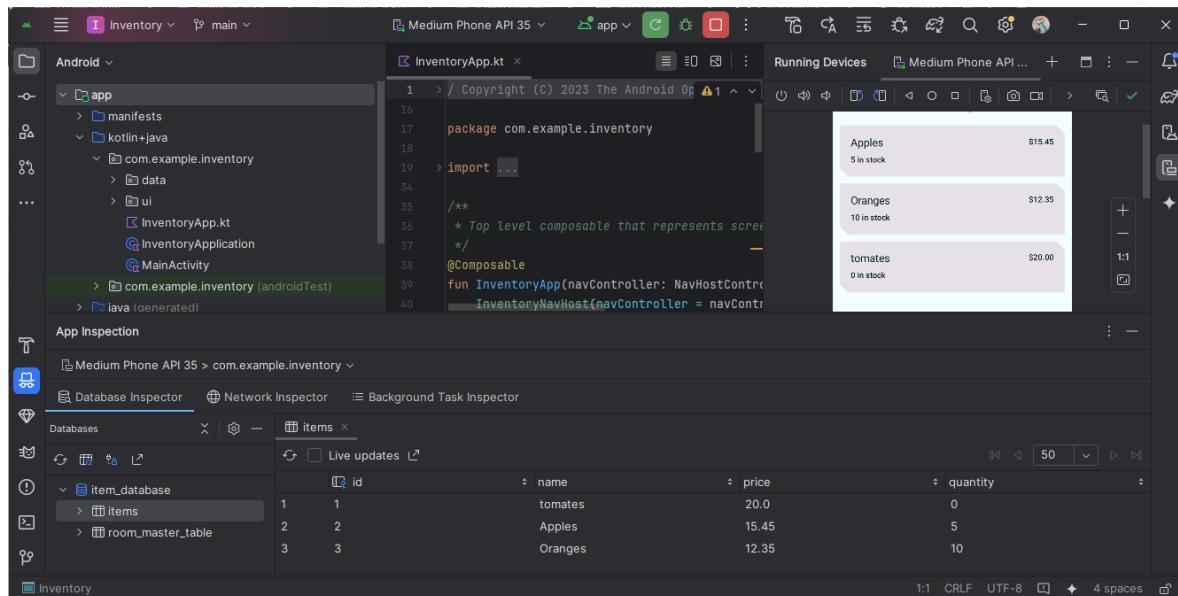
```
24 /**
25  * Database class with a singleton Instance object.
26 */
27 @Database(entities = [Item::class], version = 1, exportSchema = false)
28 abstract class InventoryDatabase : RoomDatabase() { // Clase abstracta que hereda de RoomDatabase
29
30     abstract fun itemDao(): ItemDao // Método abstracto para obtener una instancia del DAO (interface)
31
32     companion object {
33         @Volatile
34         private var _instance: InventoryDatabase? = null // Instancia Única de la base de datos
35
36         fun getDatabase(context: Context): InventoryDatabase { // Función para obtener o crear la base de datos
37             // If the Instance is not null, return it, otherwise create a new database instance.
38             return _instance ?: synchronized(lock) {
39                 // Crea un constructor para la base de datos
40                 Room.databaseBuilder(context, InventoryDatabase::class.java, name: "item_database")
41
42                     /**
43                         * Setting this option in your app's database builder means that Room
44                         * permanently deletes all data from the tables in your database when it
45                         * attempts to perform a migration with no defined migration path.
46                         */
47                     .fallbackToDestructiveMigration() // Elimina todos los datos si no se encuentra
48                     .build() // Construye la instancia de la base de datos
49                     .also { _instance = it } // Asigna la instancia creada a 'Instance' y la devuelve
50             }
51         }
52     }
53 }
```

```
26 /**
27  * Database access object to access the Inventory database
28  * (Interfaz de acceso a la base de datos para interactuar con los datos de inventario)
29  */
30 @Dao // Indica que esta es una interfaz DAO que contiene operaciones de acceso a la base de datos
31 interface ItemDao { // Define la interfaz ItemDao para gestionar operaciones CRUD sobre la tabla 'items'
32
33     @Query("SELECT * from items ORDER BY name ASC") // Consulta para obtener todos los items ordenados por nombre
34     suspend fun getAllItems(): Flow<List<Item>> // Devuelve una lista de 'items' como un flujo de datos (para manejar cambios en tiempo real)
35
36     @Query("SELECT * from items WHERE id = :id") // Consulta para obtener un item específico por su ID
37     suspend fun getItem(id: Int): Flow<Item> // Devuelve un solo item basado en el id, como un flujo de datos
38
39     // Specify the conflict strategy as IGNORE, when the user tries to add an
40     // existing Item into the database Room ignores the conflict.
41     @Insert(onConflict = OnConflictStrategy.IGNORE) // Inserta un item en la base de datos, ignorando conflictos
42     suspend fun insert(item: Item) // Operación suspensiva para insertar un item en la base de datos
43
44     @Update // Anotación que indica que este método se usa para actualizar un item en la base de datos
45     suspend fun update(item: Item) // Operación suspensiva para actualizar un item en la base de datos
46
47     @Delete // Anotación que indica que este método se usa para eliminar un item de la base de datos
48     suspend fun delete(item: Item) // Operación suspensiva para eliminar un item de la base de datos
49
50 }
```

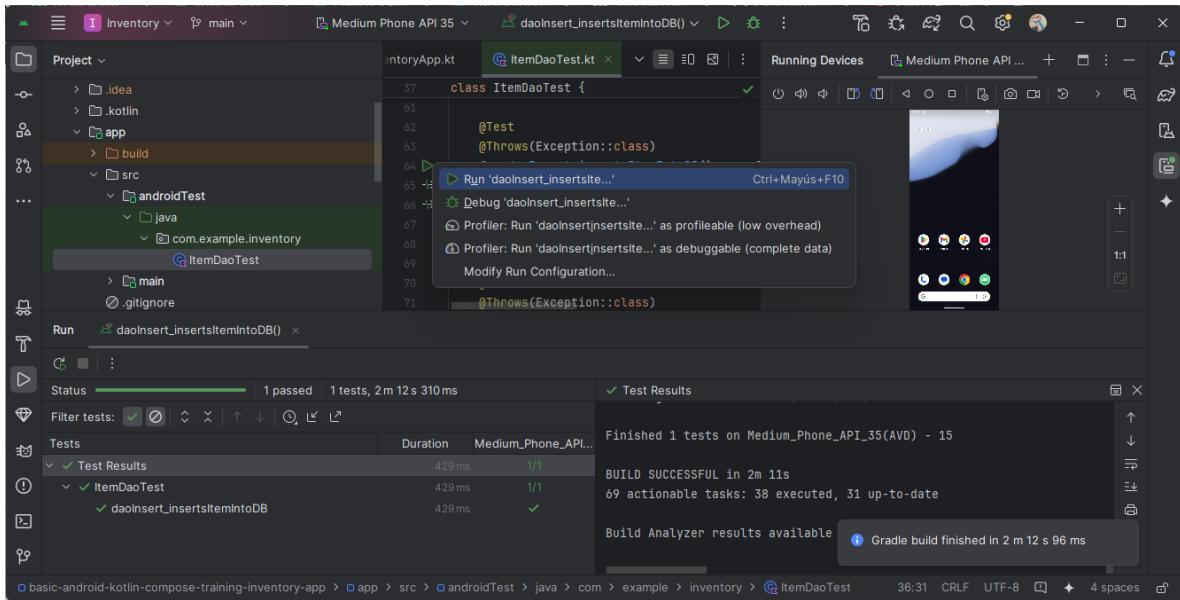
Si consultamos el contenido de la base de datos con el Inspector de bases de datos podemos ver los artículos agregados en nuestra aplicación:



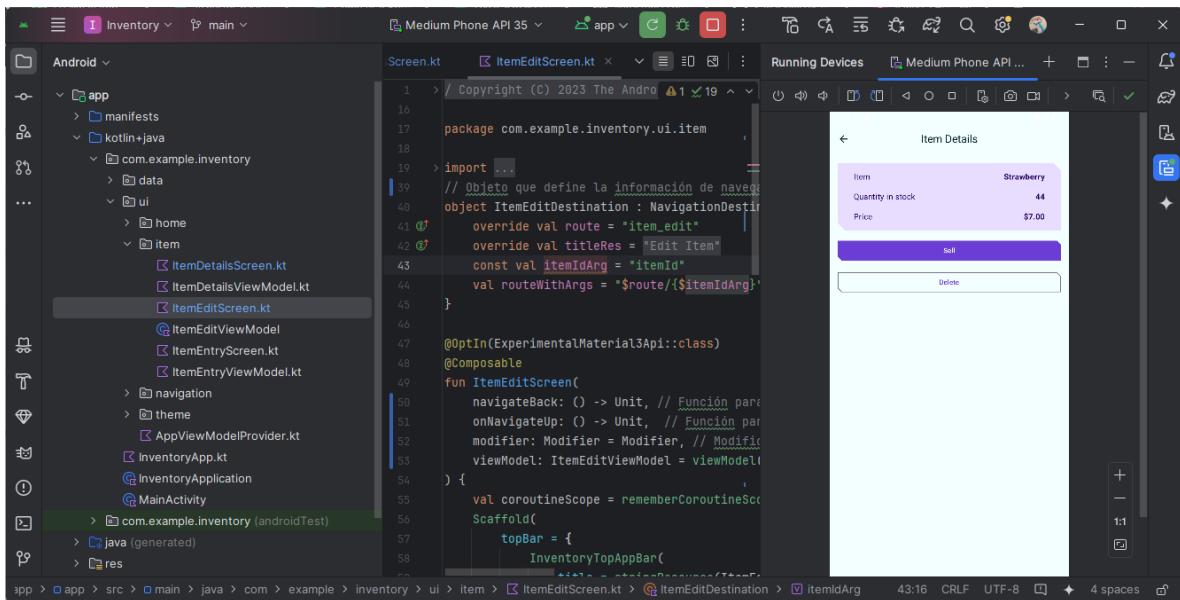
Nos vamos a Database Inspector y vemos los datos de nuestro inventario:



En cuanto a la sección de leer y actualizar datos con Room para probar la base de datos ejecutamos la prueba, la cual da exitosa:



Modificamos de forma que en el “ViewModel”, se agregó una función para recuperar un producto específico por su ID (retrieveItem) y otra función para eliminarlo (deleteItem). También se implementó la función “sellItem”, que verifica si hay unidades disponibles antes de disminuir la cantidad y actualizar el producto en la base de datos. Estas funciones se conectaron a los botones de la interfaz mediante llamadas dentro de “ItemDetailsScreen”. Finalmente, se agregó la navegación hacia atrás después de eliminar un producto, asegurando que la aplicación regrese a la lista de inventario. Con estas modificaciones, las acciones de vender y eliminar productos funcionan correctamente al seleccionar un elemento desde la interfaz:



Ahora como podemos ver, cada vez que le demos clic en sell disminuirá uno:

The screenshot shows the Android Studio interface with the project 'Inventory' open. The left sidebar shows the file structure, and the main area displays the code for 'ItemEditScreen.kt'. The code defines a navigation object for item editing, setting the route to 'item\_edit'. It also includes a scaffold for the screen with a top bar labeled 'InventoryTopAppBar'. The right side of the interface shows a preview of the 'Item Details' screen, which displays an item named 'Strawberry' with a quantity of 40 and a price of \$7.00. There are buttons for 'Sell' and 'Delete'.

```
1 > / Copyright (C) 2023 The Android Open Source Project
16
17 package com.example.inventory.ui.item
18
19 import ...
20 // Objeto que define la información de navegación
21 object ItemEditDestination : NavigationDestination {
22     override val route = "item_edit"
23     override val titleRes = R.string.Edit_Item
24     const val itemIdArg = "itemId"
25     val routeWithArgs = "$route/{$itemIdArg}"
26 }
27
28 @OptIn(ExperimentalMaterial3Api::class)
29 @Composable
30 fun ItemEditScreen(
31     navigateBack: () -> Unit, // Función para navegar hacia atrás
32     onNavigateUp: () -> Unit, // Función para navegar hacia arriba
33     modifier: Modifier = Modifier, // Modificador para el contenedor
34     viewModel: ItemEditViewModel = viewModel()
35 ) {
36     val coroutineScope = rememberCoroutineScope()
37     Scaffold(
38         topBar = {
39             InventoryTopAppBar(
40                 title = stringResource(id = R.string.Edit_Item),
41                 navigationIcon = { IconButton(onClick = navigateBack) },
42                 actions = {
43                     IconButton(onClick = onNavigateUp)
44                     IconButton(onClick = { viewModel.deleteItem() })
45                 }
46             )
47         },
48         content = {
49             ItemEditContent(
50                 itemId = itemIdArg,
51                 viewModel = viewModel,
52                 onSellClick = { viewModel.sellItem() },
53                 onDeleteClick = { viewModel.deleteItem() }
54             )
55         }
56     )
57 }
58
```

En caso de que quisiéramos eliminar un producto también se podría.

Por otra parte, en lo que respecta a la app de Bus Schedule se muestran horarios de autobuses usando Room para la base de datos y Jetpack Compose para la UI. Primero, se define la estructura de la base de datos con una entidad “BusSchedule”, luego se configura un DAO para acceder a los datos. Se implementa una instancia de base de datos con Room, y finalmente, el “ViewModel” es actualizado para gestionar y proporcionar los datos a la UI.

The screenshot shows the Android Studio interface with the project 'Bus Schedule' open. The left sidebar shows the file structure, and the main area displays the code for 'AppDatabase.kt'. The code defines an abstract class 'AppDatabase' extending 'RoomDatabase' with an entity 'BusSchedule'. It also includes a companion object with a static method 'getDatabase' that returns a database instance using 'Room.databaseBuilder'. The right side of the interface shows a preview of the 'Bus Schedule' screen, which lists bus stops with their arrival times.

```
1 > / Copyright (C) 2023 The Android Open Source Project
16
17 package com.example.busschedule.data
18
19 import ...
20
21 // Definición de la clase AppDatabase, ...
22 // La anotación @Database indica que es una base de datos
23 // @Database(entities = arrayOf(BusSchedule::class))
24 abstract class AppDatabase : RoomDatabase() {
25     // Método abstracto para acceder a la base de datos
26     abstract fun busScheduleDao(): BusScheduleDao
27
28     // Compañero de objeto que mantiene la instancia
29     companion object {
30         // Volatile asegura que el valor es ...
31         @Volatile
32         private var INSTANCE: AppDatabase? = null
33
34         // Método estático todo para obtener la instancia
35         fun getDatabase(context: Context): AppDatabase {
36             // Si ya existe una instancia, ...
37             return INSTANCE ?: synchronized(this) {
38                 // Construcción de la base de datos
39                 val db = Room.databaseBuilder(
40                     context,
41                     AppDatabase::class.java,
42                     "bus-schedule"
43                 ).build()
44                 INSTANCE = db
45                 return db
46             }
47         }
48     }
49 }
```

Así se vería la tabla de datos final:

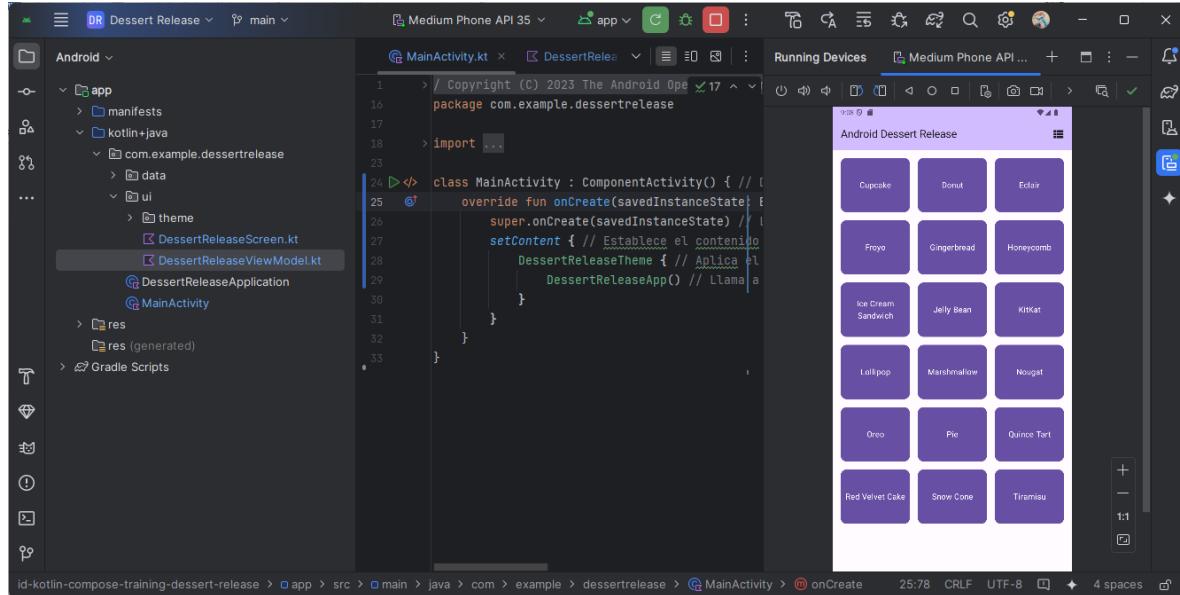
The screenshot shows the Android Studio interface. On the left, the project structure for 'Bus Schedule' is visible, showing the 'app' module with 'kotlin+java' and 'com.example.busschedule' packages. In the center, the code editor displays 'AppDatabase.kt' with Kotlin code for a database helper class. On the right, the 'Running Devices' window shows a preview of an Android application titled 'Bus Schedule' displaying a bus arrival schedule with stops like Main Street, Park Street, Maple Avenue, and Broadway Avenue, along with their arrival times. Below the code editor, the 'Database Inspector' tool is open, showing the 'app\_database' database with a 'Schedule' table containing the same data as the UI.

### Ruta de Aprendizaje 3 (Cómo almacenar datos y acceder a ellos mediante claves con DataStore)

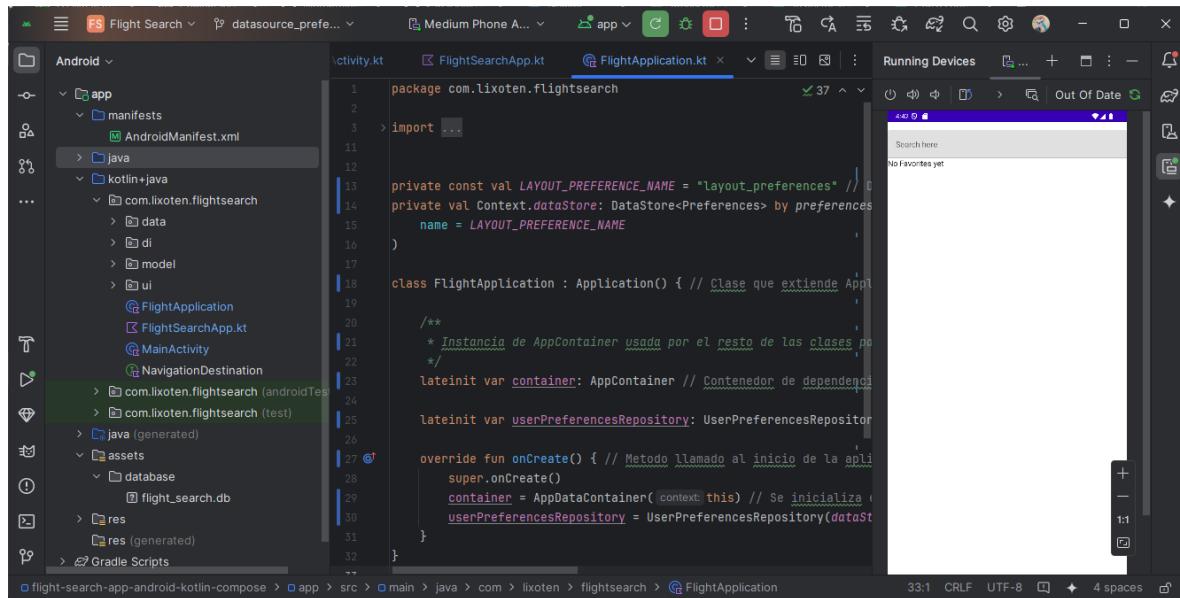
En cuanto a la sección de guardar las preferencias de forma local con DataStore el código implementa un “ViewModel” en “DessertReleaseViewModel” que gestiona el estado del layout (lineal o en cuadrícula) de la aplicación mediante un flujo reactivo (StateFlow). Utiliza “UserPreferencesRepository” para guardar y recuperar las preferencias del usuario en un almacenamiento persistente. Además, se implementa una interfaz que permite cambiar entre layouts (lineal o cuadrícula) y actualizar la UI automáticamente. La “Factory” de “ViewModel” proporciona la instancia del “ViewModel” con dependencias adecuadas, usando la clase “DessertReleaseApplication”.

The screenshot shows the Android Studio interface. On the left, the project structure for 'Dessert Release' is visible, showing the 'app' module with 'kotlin+java' and 'com.example.dessertrelease' packages. In the center, the code editor displays 'MainActivity.kt' with Kotlin code for the main activity. On the right, the 'Running Devices' window shows a preview of an Android application titled 'Android Dessert Release' displaying a list of dessert names: Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich, Jelly Bean, KitKat, Lollipop, and Marshmallow.

Si damos clic en el ícono superior derecho podemos observar cómo cambia el diseño de nuestra app. Incluso si salimos de esta se guardan las preferencias:



En cuanto al proyecto que se nos solicita de crear una app de búsqueda de vuelos, donde lo que nosotros hicimos fue permitir a los usuarios buscar vuelos disponibles desde un aeropuerto de salida, visualizar los resultados y marcar vuelos como favoritos. Para esto se utilizó una base de datos pre-poblada con información de vuelos y se almacenaron las preferencias del usuario para mejorar la experiencia en futuras sesiones.



En cuanto al diseño de pantallas el diseño de la interfaz de usuario esta se realiza completamente con Jetpack Compose. La pantalla de Búsqueda (Search Screen) contiene un campo de texto donde el usuario puede ingresar el código IATA o el

nombre de un aeropuerto de salida donde al escribir, se filtran y muestran los vuelos disponibles desde ese aeropuerto. Si el campo de búsqueda está vacío, se muestra una lista de vuelos marcados como favoritos.

En cuanto a la pantalla de Resultados de Vuelo (Flight Screen) esta muestra los detalles de los vuelos disponibles desde el aeropuerto seleccionado, donde cada vuelo tiene la opción de ser marcado o desmarcado como favorito.

La navegación entre estas pantallas se gestiona mediante Navigation Compose, permitiendo una transición fluida y manejo de argumentos entre composable.

Para la conexión a la Base de Datos la aplicación utiliza Room para manejar una base de datos local la cual contiene dos tablas principales:

- Airports: Contiene información sobre los aeropuertos disponibles.
- Favorites: Almacena los vuelos que el usuario ha marcado como favoritos.

Al iniciar la aplicación, se carga la base de datos pre-poblada con información de vuelos. Cuando un usuario marca un vuelo como favorito, se inserta un nuevo registro en la tabla de favoritos.

La funcionalidad de búsqueda se implementa mediante la observación del campo de texto en la pantalla de búsqueda: A medida que el usuario escribe, se realiza una consulta en la base de datos para encontrar vuelos que coincidan con el texto ingresado. Los resultados se actualizan en tiempo real y si no se encuentra ningún resultado, se muestra un mensaje indicando que no hay vuelos disponibles para la búsqueda realizada.

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

- Project Structure:** Shows the project tree under "Android". The "app" module contains "src/main/java/com.lixoten.flightssearch" which includes "FlightApplication.kt", "FlightSearchApp.kt", "MainActivity.kt", and "NavigationDestination.kt".
- Code Editor:** Displays the "FlightApplication.kt" file with the following code:package com.lixoten.flightssearch
import ...
private const val LAYOUT\_PREFERENCE\_NAME = "layout\_preferences"
private val Context.dataStore: DataStore<Preferences> by preferences {
 name = LAYOUT\_PREFERENCE\_NAME
}
class FlightApplication : Application() {
 lateinit var container: AppContainer
 lateinit var userPreferencesRepository: UserPreferencesRepository
 override fun onCreate() {
 super.onCreate()
 container = AppDataContainer(context = this)
 userPreferencesRepository = UserPreferencesRepository(dataStore)
 }
}
- Preview Window:** Shows a search interface with a list of airports. The list includes:
  - MAD Adolfo Suárez Madrid-Barajas Airport
  - ALC Alicante Airport
  - AMS Amsterdam Airport Schiphol
  - ATH Athens International Airport
  - TXL Berlin-Tegel Airport
  - BRU Brussels Airport
  - CPH Copenhagen Airport
  - DUB Dublin Airport
  - DUS Düsseldorf International Airport
  - L EuroAirport Basel Mulhouse Freiburg
  - GPO Francisco Sá Carneiro Airport
  - FRA Frankfurt Airport
  - GVA Genève Airport
  - HAM Hamburg Airport
  - HEL Helsinki Airport
  - HER Heraklion Airport N. Kazantzakis
  - LIS Humberto Delgado Airport
  - BON Josep Tarradellas Barcelona El Prat Airport

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

- Project Structure:** The left sidebar shows the project structure under "Android". It includes the "app" module with its manifest file "AndroidManifest.xml", a "java" directory containing "com.lixoten.flightssearch" with sub-directories "data", "di", "model", and "ui", and files "FlightApplication.kt", "FlightSearchApp.kt", and "MainActivity.kt". There are also "test" and "generated" directories.
- Code Editor:** The main editor window displays the "FlightApplication.kt" file. The code initializes an AppContainer and a UserPreferencesRepository, and overrides the onCreate() method to initialize these components.
- Running Devices:** The right sidebar shows a list of favorite airports. The list includes:
  - MAD Adolfo Suárez Madrid-Barajas Airport
  - OPO Francisco Sá Carneiro Airport
  - MAD Adolfo Suárez Madrid-Barajas Airport
  - ARN Stockholm Arlanda Airport
  - MAD Adolfo Suárez Madrid-Barajas Airport
  - WAW Warsaw Chopin Airport
  - MAD Adolfo Suárez Madrid-Barajas Airport
  - MRS Marseille Provence Airport
  - MAD Adolfo Suárez Madrid-Barajas Airport
  - BGY Milan Bergamo Airport
  - MAD Adolfo Suárez Madrid-Barajas Airport
  - VIE Vienna International Airport
  - MAD Adolfo Suárez Madrid-Barajas Airport
  - SVO Sheremetyevo - A.S. Pushkin International airport
  - DUB Dublin Airport
  - MAD Adolfo Suárez Madrid-Barajas Airport
  - SOF Sofia Airport
  - MAD Adolfo Suárez Madrid-Barajas Airport
  - CPH Copenhagen Airport
  - MAD Adolfo Suárez Madrid-Barajas Airport
  - BRU Brussels Airport
  - MAD Adolfo Suárez Madrid-Barajas Airport
- Bottom Status Bar:** Shows the file name "FlightApplication.kt", line number "33:1", and encoding "CRLF UTF-8".

En cuanto a la gestión de favoritos los usuarios pueden marcar o desmarcar vuelos como favoritos directamente desde la pantalla de resultados donde al marcar un vuelo como favorito, se inserta un nuevo registro en la tabla de favoritos de la base de datos. Estos vuelos favoritos se muestran en la pantalla principal cuando el campo de búsqueda está vacío, permitiendo un acceso rápido a los vuelos preferidos del usuario.

En el almacenamiento de preferencias la aplicación utiliza DataStore Preferences para almacenar las preferencias del usuario, como el último aeropuerto buscado: Al realizar una búsqueda, el texto ingresado se guarda en las preferencias del usuario. Cuando la aplicación se reinicia, el campo de búsqueda se pre-puebla con el último texto ingresado, mejorando la experiencia del usuario al recordar sus acciones anteriores.

The screenshot shows the Android Studio interface with the FlightSearch project open. The code editor displays the `FlightApplication.kt` file, which initializes the application context and dependency container. The `Running Devices` window on the right lists various airports, including Adolfo Suárez Madrid-Barajas Airport, Francisco Sá Carneiro Airport, Stockholm Arlanda Airport, Warsaw Chopin Airport, and Copenhagen Airport, many of which have a red heart icon indicating they are favorite locations.

```
package com.lixoten.flightssearch
import ...
private const val LAYOUT_PREFERENCE_NAME = "layout_preferences"
private val Context.dataStore: DataStore<Preferences> by preferences(name = LAYOUT_PREFERENCE_NAME)
class FlightApplication : Application() {
    override fun onCreate() {
        super.onCreate()
        container = AppDataContainer(context = this)
        userPreferencesRepository = UserPreferencesRepository(dataStore)
    }
}
```

This screenshot is similar to the one above, showing the same code in `FlightApplication.kt`. However, the `Running Devices` window has been filtered to show only a subset of airports, specifically those starting with 'MAD' or 'DUB'. A search bar at the top of the list allows for further filtering.

```
package com.lixoten.flightssearch
import ...
private const val LAYOUT_PREFERENCE_NAME = "layout_preferences"
private val Context.dataStore: DataStore<Preferences> by preferences(name = LAYOUT_PREFERENCE_NAME)
class FlightApplication : Application() {
    override fun onCreate() {
        super.onCreate()
        container = AppDataContainer(context = this)
        userPreferencesRepository = UserPreferencesRepository(dataStore)
    }
}
```

Si hacemos una consulta a nuestra base de datos podemos ver los aeropuertos que tenemos en nuestra base y los que hemos agregado a favoritos:

The screenshot shows the Android Studio interface with the project 'Flight Search' open. The code editor displays the 'FlightApplication.kt' file, which initializes an application context and a user preferences repository. Below the code editor is the Database Inspector tool, which is connected to a SQLite database named 'flight\_database'. A query 'select \* from airport;' is run, and the results are displayed in a table:

	id	name	iata_code	passengers
1	1	Francisco Sá Carneiro Airport	OPO	5053134
2	2	Stockholm Arlanda Airport	ARN	7494765
3	3	Warsaw Chopin Airport	WAW	18860000
4	4	Marseille Provence Airport	MRS	10151743
5	5	Milan Bergamo Airport	BGY	3833063
6	6	Vienna International Airport	VIE	7812938

This screenshot is nearly identical to the one above, showing the same project structure and code in 'FlightApplication.kt'. The Database Inspector shows the results of the query 'select \* from favorite;':

	id	departure_code	destination_code
1	1	MAD	OPO
2	2	MAD	DUB

El MainActivity.kt es el punto de entrada de la aplicación que configura el tema y establece el contenido principal. FlightSearchApp.kt, por su parte, define la navegación y las pantallas principales de la aplicación. Mientras que en ui/screens/search/SearchScreen.kt se implementa la pantalla de búsqueda y la lógica asociada. En ui/screens/flight\_screen/FlightScreen.kt ya se implementa la pantalla de resultados de vuelo y la gestión de favoritos, y en data está el contenido de las clases relacionadas con la base de datos y las preferencias del usuario.

**Nota:** Los comentarios están dentro de los códigos en el repositorio de github.

## Unidad 7: WorkManager

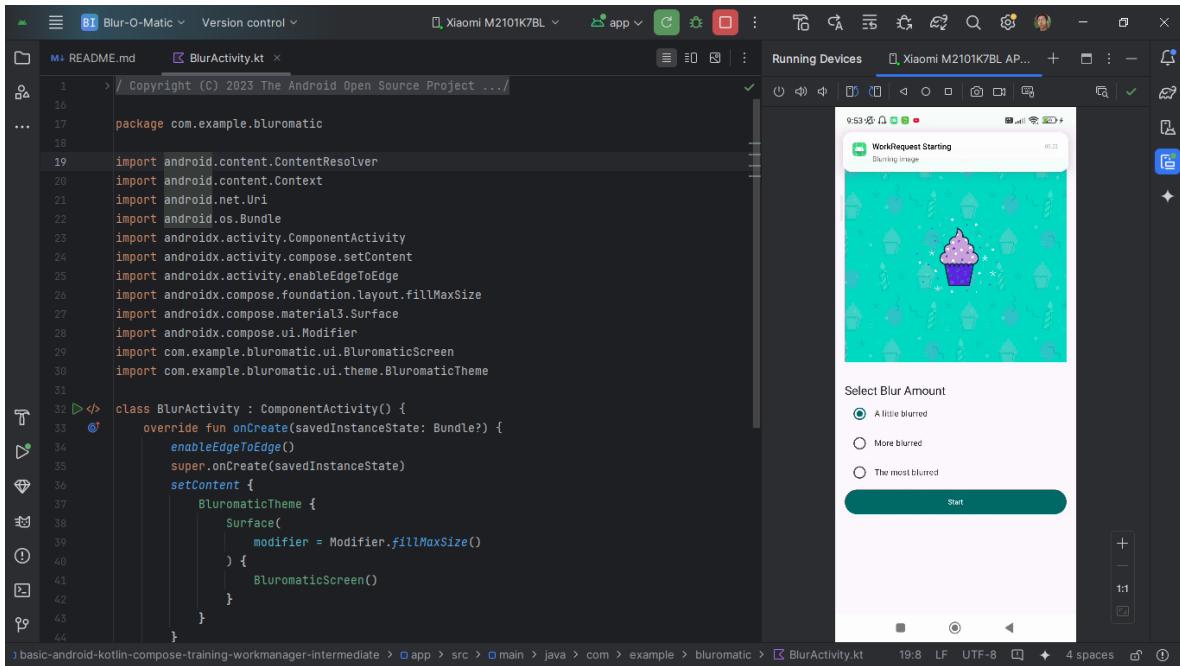
### Ruta de Aprendizaje 1 (Cómo programar tareas con WorkManager)

#### Trabajo en segundo plano con WorkManager

Este codelab se enfoca en la integración de la biblioteca WorkManager en una aplicación de Android desarrollada con Kotlin y Jetpack Compose, denominada Blur-O-Matic, donde la aplicación tiene como funcionalidad principal permitir al usuario seleccionar una imagen, aplicarle un nivel de desenfoque deseado y guardar el resultado en un archivo. En el estado inicial del codelab, la interfaz de usuario ya está construida, presentando botones para elegir la intensidad del desenfoque y un botón de inicio para el proceso.

El archivo app/build.gradle.kts incluye la dependencia necesaria para la biblioteca WorkManager, específicamente implementation("androidx.work:work-runtime-ktx:2.8.1"), que habilita la funcionalidad de WorkManager y sus extensiones Kotlin para una integración más fluida con corrutinas. Dentro de la carpeta java/com/example/bluromatic/workers/, se encuentra el archivo BlurWorker.kt, que implementa la clase CoroutineWorker y su método doWork(), donde reside la lógica para llevar a cabo el desenfoque de la imagen de entrada y el almacenamiento de la imagen resultante. En la carpeta java/com/example/bluromatic/data/, el archivo WorkManagerBluromaticRepository.kt define una clase que actúa como intermediario con WorkManager. Esta clase contiene una instancia de WorkManager y proporciona métodos como applyBlur(blurLevel: Int) para construir y poner en cola las OneTimeWorkRequest destinadas al BlurWorker.

El archivo java/com/example/bluromatic/viewmodel/BlurViewModel.kt contiene la lógica para gestionar el estado de la interfaz de usuario, invocando métodos como applyBlur en respuesta a las acciones del usuario. El archivo java/com/example/bluromatic/ui/BluromaticScreen.kt define la interfaz de usuario de la aplicación utilizando Jetpack Compose, incluyendo los elementos para la selección del nivel de desenfoque y el botón de inicio, interactuando con el BlurViewModel para enviar eventos y observar el estado. Finalmente, en la carpeta java/com/example/bluromatic/util/ está contenido el archivo WorkerUtils.kt, que agrupa métodos de utilidad utilizados por los Workers, como la visualización de notificaciones para informar al usuario sobre el progreso del trabajo y la función para guardar un objeto Bitmap en un archivo. Por su parte, el archivo java/com/example/bluromatic/Constants.kt define constantes utilizadas a lo largo de la aplicación, como nombres de archivos y claves para la transferencia de datos entre Workers.

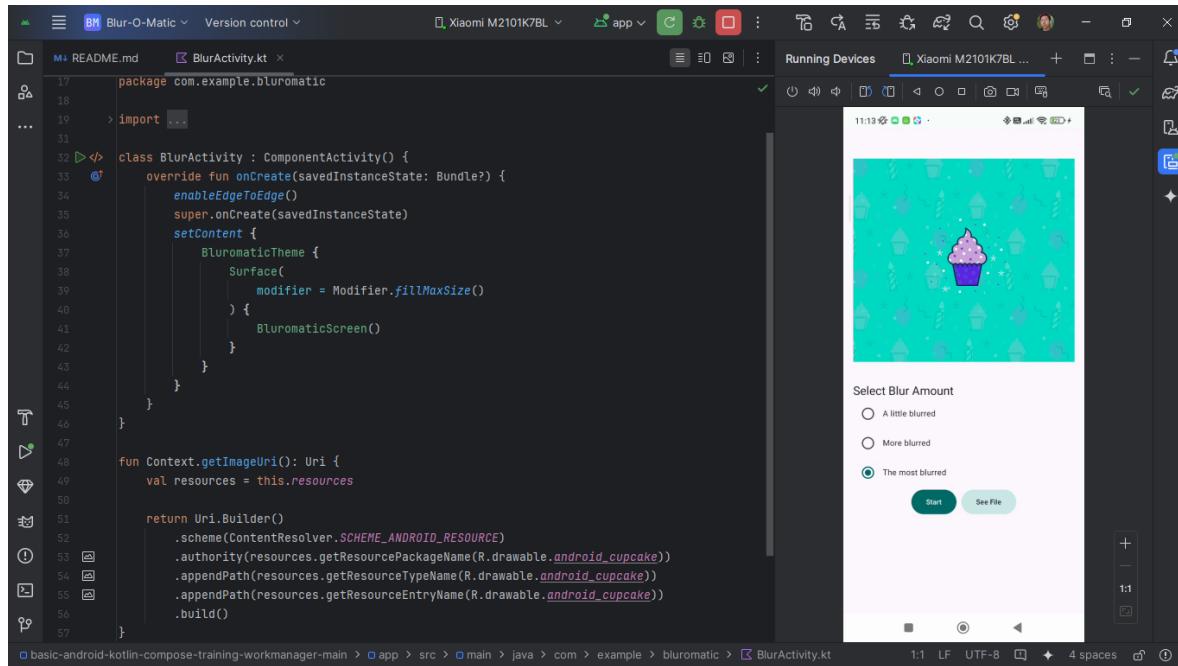


## Pruebas y WorkManager avanzados

Esta sección se centra en cómo etiquetar las `WorkRequest` y cómo utilizar el estado del trabajo en segundo plano para actualizar la interfaz de usuario de la aplicación `Blur-O-Matic`. Se explica que para rastrear el estado de la tarea de guardar la imagen final (`SavelfImageToFileWorker`), es útil asignarle una etiqueta. Esto permite consultar el estado de todas las `WorkRequest` con esa etiqueta, lo cual es preferible al uso de IDs individuales, especialmente si el usuario realiza la operación de desenfoque varias veces.

En la capa de la interfaz de usuario (`ui/BluromaticScreen.kt`), se muestra cómo se utiliza el `blurUiState` para determinar qué elementos componibles se deben mostrar. Un bloque `when` se utiliza para renderizar diferentes partes de la UI basadas en el estado actual (Default: muestra el botón "Start"; Loading: muestra el botón "Cancel Work" y un indicador de progreso; Complete: inicialmente muestra el botón "Start").

La última parte de esta sección se enfoca en cómo mostrar un botón "See File" una vez que la imagen desenfocada ha sido guardada. Esto se logra modificando el estado `BlurUiState.Complete` para que contenga la URI de la imagen guardada. En el `ViewModel` (`ui/BlurViewModel.kt`), dentro de la transformación `map`, se recupera la URI de la imagen guardada de los `outputData` del `WorkInfo` utilizando la clave `KEY_IMAGE_URI`. Luego, el estado `BlurUiState.Complete` se actualiza para incluir esta URI. En la interfaz de usuario (`ui/BluromaticScreen.kt`), dentro del bloque `when` para `BlurUiState.Complete`, se añade un botón "See File" que recibe una lambda `onClick` que llama a una función `showBlurredImage()` para mostrar la imagen utilizando su URI.



## Práctica: App de Water Me!

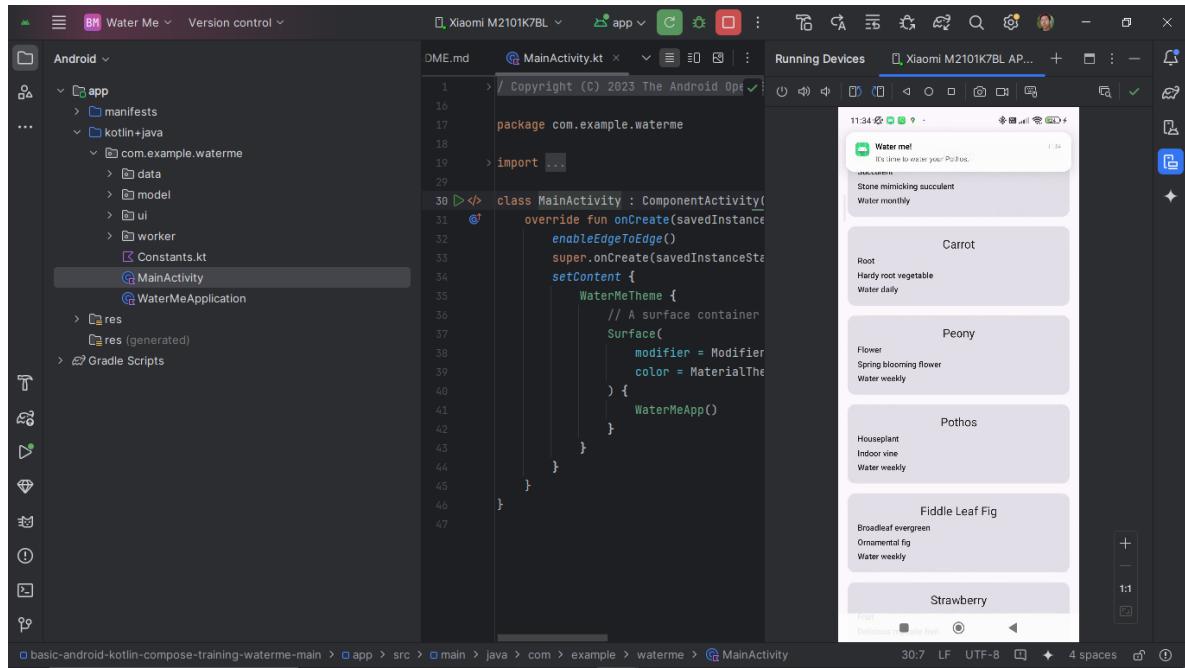
En este codelab nos centramos en la implementación de la funcionalidad de programar notificaciones de recordatorio de riego utilizando la biblioteca WorkManager. El crear y mostrar una notificación reside en la clase `WaterReminderWorker.kt`, que extiende `CoroutineWorker`. Dentro del método `doWork()` de esta clase, se extrae el nombre de la planta de los datos de entrada y se utiliza para construir y mostrar la notificación al usuario. La función `scheduleReminder()` dentro del archivo `WorkManagerWaterRepository.kt` es la encargada de orquestar la creación de una `OneTimeWorkRequest`, configurándola para que ejecute el `WaterReminderWorker` en un momento futuro especificado por el usuario, y proporcionándole la información necesaria, que en este caso es el nombre de la planta.

El proceso para implementar `scheduleReminder()` implica varios pasos clave. Como el crear un objeto `Data` utilizando un `Data.Builder`. Este objeto actuará como un contenedor para los datos que se pasarán al `WaterReminderWorker`. En este caso, el dato crucial es el nombre de la planta, que se almacenará en el objeto `Data` utilizando la clave definida como `WaterReminderWorker.nameKey` y el valor proporcionado por el parámetro `plantName` de la función `scheduleReminder()`. Una vez que los datos de entrada están preparados, se construye una `OneTimeWorkRequest`. Esta solicitud de trabajo especificará que el `WaterReminderWorker` es el componente que se debe ejecutar. Al construir la solicitud, se indica el retraso antes de la ejecución, utilizando los parámetros `duration` y `unit` que también se pasan a `scheduleReminder()`. Además, se adjunta el objeto `Data` creado previamente como los datos de entrada para esta `WorkRequest`. Finalmente, se utiliza el método `enqueueUniqueWork()` de la instancia de

WorkManager. Este método es importante para asegurar que no se creen múltiples recordatorios idénticos para la misma planta con la misma duración.

En resumen, en WaterReminderWorker.kt está la definición del trabajador encargado de mostrar la notificación. Este trabajador, al extender CoroutineWorker, implementa el método doWork(), que constituye el núcleo de su funcionalidad. Dentro de este método, se recupera el nombre de la planta a partir de los datos de entrada proporcionados mediante la clave nameKey. Tras una verificación para asegurar que el nombre de la planta no sea nulo, se invoca una función, makeStatusNotification(), para construir y presentar la notificación al usuario, utilizando el nombre de la planta en el mensaje. Finalmente, el método retorna Result.success() para indicar a WorkManager que la tarea se ha completado de manera exitosa.

Por otro lado, en WorkManagerWaterRepository.kt se halla la implementación de la función scheduleReminder(), la cual juega un papel crucial en la programación de la ejecución del WaterReminderWorker. Esta función recibe la duración y la unidad de tiempo del recordatorio, así como el nombre de la planta. Inicialmente, se crea un objeto Data utilizando un Data.Builder, donde se almacena el nombre de la planta utilizando la clave WaterReminderWorker.nameKey. Acto seguido, se construye una OneTimeWorkRequest, especificando que WaterReminderWorker es el trabajador a ejecutar y configurando el retraso inicial mediante setInitialDelay() con la duración y unidad de tiempo proporcionadas. Los datos de entrada para el trabajador se asignan utilizando setInputData() con el objeto Data creado. Finalmente, la WorkRequest se programa utilizando workManager.enqueueUniqueWork(), proporcionando un nombre único para el trabajo (basado en el nombre de la planta y la duración), una política para trabajos existentes con el mismo nombre (ExistingWorkPolicy.REPLACE) y la waterReminderRequest construida. Este mecanismo asegura que el WaterReminderWorker se ejecute en el momento programado, con la información necesaria para personalizar la notificación, y evita la creación de recordatorios duplicados para la misma planta con la misma configuración de tiempo.



**Nota:** Los códigos con comentarios se encuentran subidos en [github](#).

## Unidad 8: Compose con Views

### Ruta de Aprendizaje 1 (Views de Android y Compose en Views)

#### Cómo compilar una app para Android con objetos View

El objetivo de esta sección es construir la interfaz de usuario para un diálogo que permita al usuario ingresar el tipo de zumo que ha consumido. Tradicionalmente, en el desarrollo de aplicaciones Android con Views, la interfaz de usuario se define en archivos de layout XML. Estos archivos, ubicados en el directorio res > layout, contienen la estructura visual de la pantalla o un componente de la interfaz, utilizando elementos XML que representan los componentes de la IU (Views) y los contenedores que los organizan (Layouts o ViewGroups).

El primer paso por el que nos guía el codelab es crear un nuevo archivo de recurso de layout XML llamado fragment\_entry\_dialog.xml. El elemento raíz de este layout es un ConstraintLayout. ConstraintLayout es un tipo de ViewGroup que ofrece una gran flexibilidad para posicionar y dimensionar las Views secundarias mediante la definición de restricciones relativas a otros componentes o a líneas guía dentro del layout.

Dentro del ConstraintLayout, se definen tres elementos Guideline. Las Guidelines son líneas visuales no renderizadas que ayudan a alinear y posicionar otros elementos de la interfaz. En este caso, se crean una guía vertical a 16dp del borde izquierdo (guideline\_left), una guía vertical en el medio del layout (guideline\_middle, utilizando un porcentaje del 50%), y una guía horizontal a 16dp del borde superior (guideline\_top). Estas guías actúan como márgenes o puntos de referencia para el siguiente elemento. Además, también se añade un elemento TextView con el ID header\_title. Este TextView se utiliza para mostrar el título del diálogo, que se establece mediante el atributo android:text con el valor @string/juice\_type (una cadena definida en los recursos de la aplicación). El ancho del TextView se establece en 0dp, lo que indica que su ancho será determinado por las restricciones de layout que se definan. En cuanto a la apariencia del texto esta se define mediante el atributo android:textAppearance con el estilo @style/TextAppearance.MaterialComponents.Headline5.

Finalmente, se definen las restricciones para el header\_title dentro del ConstraintLayout. La parte superior del TextView se restringe a la parte inferior de la guideline\_top (app:layout\_constraintTop\_toBottomOf="@+id/guideline\_top"). El borde final del TextView se restringe al inicio de la guideline\_middle (app:layout\_constraintEnd\_toStartOf="@+id/guideline\_middle"), y el borde inicial del TextView se restringe al inicio de la guideline\_left (app:layout\_constraintStart\_toStartOf="@+id/guideline\_left"). Estas restricciones posicionan el título en la parte superior del diálogo, con márgenes definidos por las guías izquierda y central.

La siguiente parte del codelab introduce el concepto de Fragment. Un Fragment es un componente reutilizable que representa una porción de la interfaz de usuario en una Activity. Para alojar el layout fragment\_entry\_dialog, se crea una nueva clase llamada EntryDialogFragment que extiende BottomSheetDialogFragment. BottomSheetDialogFragment es una subclase de DialogFragment que muestra un diálogo que se desliza desde la parte inferior de la pantalla.

Dentro de EntryDialogFragment, se implementa el método onCreateView(). Este método es responsable de inflar el layout XML y devolver la View resultante. En lugar de la implementación predeterminada, se utiliza FragmentEntryDialogBinding.inflate(inflater, container, false).root para inflar el layout fragment\_entry\_dialog.xml utilizando la característica de View Binding. View Binding genera automáticamente una clase de enlace (en este caso, FragmentEntryDialogBinding) que facilita el acceso a las Views definidas en el XML de forma segura y eficiente.

También se menciona la creación de una instancia de EntryViewModel para proporcionar datos a la IU, y se implementa el método onViewCreated(), que se llama después de que la View del Fragment ha sido creada.

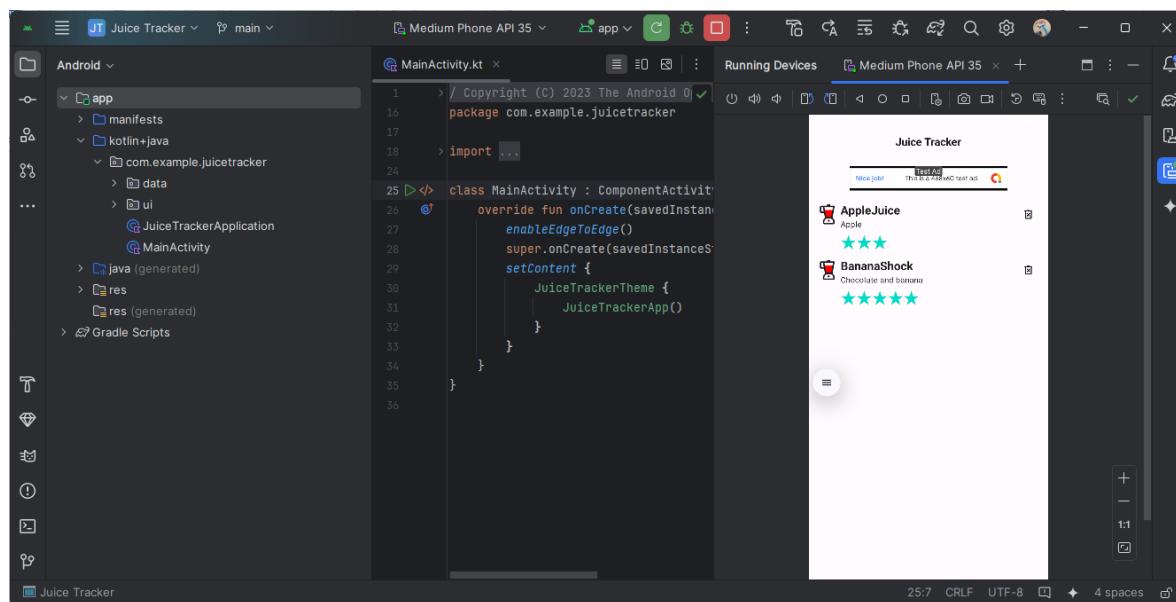
Finalmente, se aborda cómo iniciar el EntryDialogFragment utilizando el componente de Navigation. En el OnClickListener de un Floating Action Button (FAB) en otro Fragment (TrackerFragment), se utiliza el método findNavController().navigate() para navegar a un destino definido en el gráfico de navegación. Se utiliza una clase generada automáticamente (TrackerFragmentDirections) para obtener la acción de navegación hacia el EntryDialogFragment. También se muestra cómo pasar datos (en este caso, el ID de un zumo) al EntryDialogFragment al navegar desde un elemento de una lista.

En resumen, econtramos en el archivo fragment\_entry\_dialog.xml la definición estructural de la interfaz de usuario de este diálogo. El layout raíz es un androidx.constraintlayout.widget.ConstraintLayout, dentro del cual se definen elementos androidx.constraintlayout.widget.Guideline que actúan como líneas de referencia para el posicionamiento de otros componentes. Estas guías, identificadas por sus IDs (guideline\_left, guideline\_middle, guideline\_top), se sitúan mediante atributos como app:layout\_constraintGuide\_begin y app:layout\_constraintGuide\_percent, además de android:orientation. Asimismo, se declara un TextView con el ID header\_title, encargado de mostrar el título del diálogo, cuyo texto se establece mediante una referencia a un recurso de cadena (@string/juice\_type) y cuya apariencia se define a través de un estilo (@style/TextAppearance.MaterialComponents.Headline5). La posición de este TextView dentro del ConstraintLayout se determina mediante restricciones que referencian los IDs de las Guidelines, utilizando atributos como app:layout\_constraintTop\_toBottomOf, app:layout\_constraintEnd\_toStartOf, y app:layout\_constraintStart\_toStartOf.

En el archivo EntryDialogFragment.kt, se implementa la clase EntryDialogFragment, la cual es la encarga de extender lo que es com.google.android.material.bottomsheet.BottomSheetDialogFragment. Dentro de esta clase, el método onCreateView() se encarga de inflar el layout fragment\_entry\_dialog.xml utilizando la funcionalidad de View Binding, a través de la clase generada FragmentEntryDialogBinding. Se declara una variable privada para almacenar la instancia de la vinculación (\_binding) y una propiedad para acceder a ella de forma segura (binding). El método onDestroyView() se encarga de liberar la instancia de la vinculación cuando la vista del Fragment se destruye.

Por otro lado, en el archivo TrackerFragment.kt, se encuentra la lógica para iniciar la navegación hacia el EntryDialogFragment. Dentro del OnClickListener asociado al Floating Action Button (FAB), se utiliza el NavController (obtenido mediante findNavController()) para navegar a la acción definida en el gráfico de navegación que conduce al diálogo de entrada. Esta acción se genera automáticamente mediante Navigation Safe Args (TrackerFragmentDirections.actionTrackerFragmentToEntryDialogFragment()).

Finalmente, si la aplicación permite la edición de elementos de la lista de zumos, el archivo JuiceListAdapter.kt contendrá la implementación del ListAdapter y la lógica para manejar los clics en los elementos. Al detectar un clic en el botón de edición de un elemento, se invocaría una lambda que, a su vez, utilizaría el NavController desde el TrackerFragment para navegar al EntryDialogFragment, pasando el ID del zumo seleccionado como argumento a través de la acción de navegación generada por Safe Args (TrackerFragmentDirections.actionTrackerFragmentToEntryDialogFragment(drink.id)).



## Cómo agregar Compose a una app basada en objetos View

La aplicación de Juice Tracker que vimos en el apartado anterior ya permite a los usuarios agregar información sobre sus zumos (nombre, descripción, color, calificación) a una base de datos utilizando la biblioteca Room. El objetivo de este codelab es reemplazar la implementación de la lista de zumos, que originalmente estaba basada en el sistema de Views de Android, con una implementación utilizando Jetpack Compose.

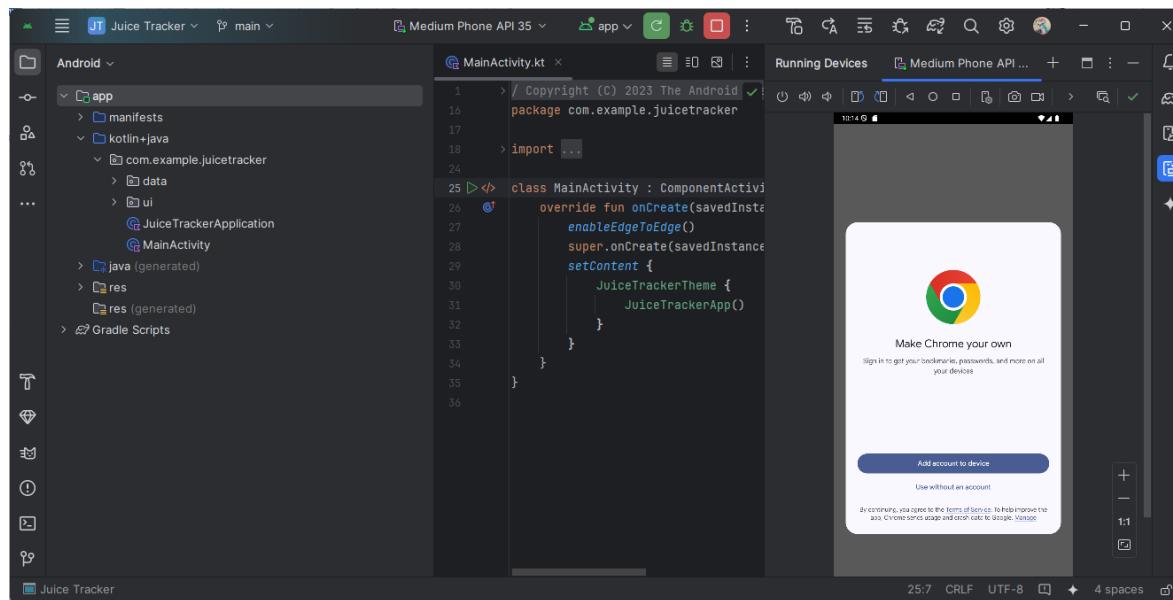
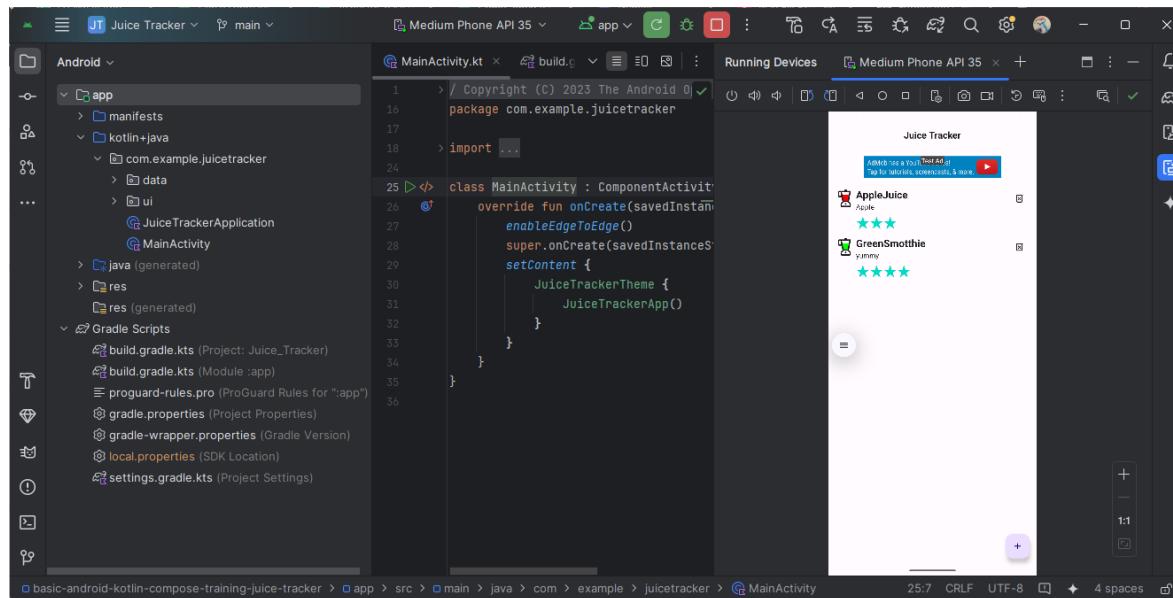
Primero, agregamos la biblioteca de Jetpack Compose al proyecto. Esto se realiza modificando el archivo build.gradle.kts a nivel de la aplicación. Dentro del bloque buildFeatures, se añade la línea compose = true para habilitar la compatibilidad con Compose en el módulo. También se añade un bloque composeOptions para especificar la versión del compilador de Kotlin que se utilizará (kotlinCompilerExtensionVersion = "1.5.1"). Finalmente, en la sección dependencies, se implementan las dependencias necesarias para integrar Compose en una aplicación basada en Views, incluyendo las dependencias para activity-compose, material3 (la biblioteca de componentes de diseño de Compose) y accompanist-themeadapter-material3 (para la compatibilidad de temas). También se añade la dependencia debugImplementation("androidx.compose.ui:ui-tooling") para herramientas de desarrollo de Compose.

Una vez configurado el proyecto para usar Compose, se introduce un ComposeView. Un ComposeView es una View de Android especial que puede hospedar contenido de la interfaz de usuario de Jetpack Compose dentro de la jerarquía de Views tradicional. Para integrar Compose en la lista existente, se modifica el JuiceListAdapter.kt. Se elimina cualquier referencia a ListItemBinding y, en la clase JuiceListViewHolder, se reemplaza binding.root por una instancia de ComposeView. En el método onCreateViewHolder(), se actualiza la función para que cree y devuelva un JuiceListViewHolder que ahora recibe un ComposeView en su constructor. También se eliminan las variables privadas y todo el código dentro de la función bind() del JuiceListViewHolder, ya que la lógica de enlace de datos se manejará con Compose. Finalmente, se elimina el archivo de layout XML list\_item.xml, ya que su funcionalidad será reemplazada por código Compose.

Por otra parte, se define una función @Composable llamada ListItem que toma un objeto Juice y dos lambdas para las acciones de editar y borrar. Dentro de ListItem, se llama a otros elementos componibles más pequeños: Juicelcon (para mostrar un ícono coloreado del zumo), JuiceDetails (para mostrar el nombre, la descripción y la calificación del zumo) y DeleteButton (para un botón de borrar). En este codelab se proporcionan detalles sobre la creación de estos elementos componibles, incluyendo cómo utilizar Box para superponer iconos en Juicelcon, cómo usar Column y Text para los detalles del zumo en JuiceDetails, y cómo implementar un sistema de calificación visual (RatingDisplay) utilizando iconos de estrella. También se crea un DeleteButton con un ícono. Se utilizan modificadores (Modifier) para

controlar el diseño, el peso y la alineación de estos elementos dentro de un Row en el ListItem principal.

Finalmente, en la función bind() del JuiceListViewHolder, se utiliza el ComposeView creado en onCreateViewHolder(). Se llama al método setContent del ComposeView, y dentro de su lambda, se invoca la función @Composable ListItem, pasándole el objeto Juice actual y las lambdas para las acciones onEdit y onDelete. Se aplican modificadores al ListItem para que ocupe todo el ancho (fillMaxWidth()), añada un padding y sea clickable para activar la acción de edición.



## **Ruta de Aprendizaje 2 (Views en Compose)**

### **Usa la interoperabilidad con objetos View en Compose**

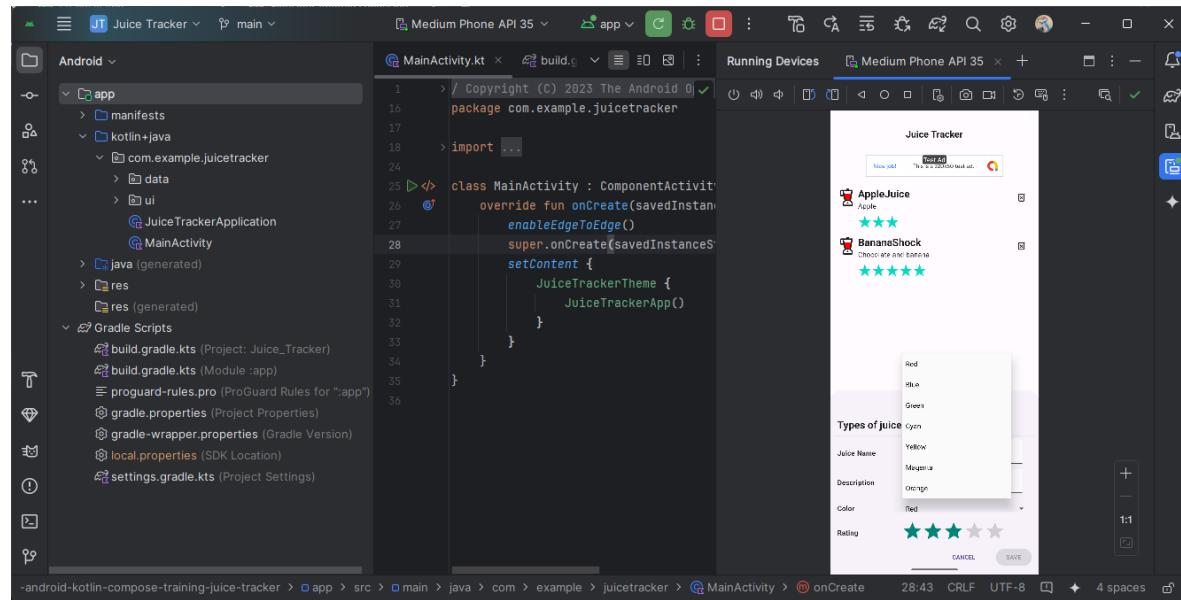
Este codelab se centra en completar el diálogo de entrada de zumo que vimos anteriormente añadiendo selectores para el color y la calificación, y también en la implementación de un banner de anuncios utilizando la biblioteca de Servicios de Play dentro de una aplicación Jetpack Compose.

Para el selector de color, dado que Compose no tiene un equivalente directo al Spinner de Android Views, se utiliza un enfoque de interoperabilidad. Se crea un nuevo archivo `ColorSpinnerRow.kt` dentro del directorio `bottomsheet`. Dentro de este archivo, se define una clase `SpinnerAdapter` que implementa `AdapterView.OnItemSelectedListener`. Este adaptador gestiona las devoluciones de llamada cuando se selecciona un elemento del Spinner, actualizando el color seleccionado en la interfaz de usuario. Dentro de `ColorSpinnerRow.kt`, también se define un elemento `@Composable` llamado `ColorSpinnerRow`. Este composable recibe la posición actual del color seleccionado y una lambda `onColorChange`. Utiliza el composable `AndroidView` para integrar un Spinner de Android Views dentro de la interfaz de Compose. Se configura un `ArrayAdapter` para proporcionar las opciones de color al Spinner, utilizando un array de strings obtenidos de la enumeración `JuiceColor`. La devolución de llamada `update` del `AndroidView` se utiliza para configurar el adaptador del Spinner, establecer la selección inicial y adjuntar el `SpinnerAdapter` para manejar los eventos de selección. Finalmente, el `ColorSpinnerRow` se integra en el `SheetForm` composable dentro de `EntryBottomSheet.kt`.

Para la entrada de calificación, se crea un nuevo archivo `RatingInputRow.kt` en el directorio `bottomsheet`. Se define un elemento `@Composable` llamado `RatingInputRow` que recibe la calificación actual y una lambda `onRatingChange`. Similar al selector de color, se utiliza `AndroidView` para integrar un `RatingBar` de Android Views. En la lambda factory del `AndroidView`, se crea una instancia de `RatingBar` y se configura el `stepSize` para que las calificaciones sean siempre números enteros. En la lambda `update`, se establece la calificación inicial de la `RatingBar` y se adjunta un `OnRatingBarChangeListener` para que cuando el usuario cambie la calificación, se invoque la lambda `onRatingChange`, actualizando el estado en la interfaz de usuario. Este `RatingInputRow` se añade también al `SheetForm` composable en `EntryBottomSheet.kt`.

Finalmente, el codelab introduce la implementación de un banner de anuncios. Se crea un nuevo archivo `AdBanner.kt` en el paquete `homescreen`. Dentro, se define un elemento `@Composable` llamado `AdBanner`. Este composable utiliza `AndroidView` para integrar una `AdView` de los Servicios de Play. En la lambda factory, se crea una instancia de `AdView`, se establece su tamaño a `AdSize.BANNER` y se asigna un ID de bloque de anuncios de prueba. En la lambda `update`, se carga un anuncio utilizando `AdRequest.Builder().build()`. Para habilitar los anuncios, se añade la

dependencia de los Servicios de Play al archivo build.gradle.kts de la aplicación y se agrega una etiqueta <meta-data> con el ID de la aplicación de anuncios al AndroidManifest.xml. El composable AdBanner se coloca en la JuiceTrackerApp.kt antes de la lista de zumos (JuiceTrackerList).



**Nota:** Los códigos con comentarios se encuentran subidos en [github](#).

Cabe aclarar que se han subido los proyectos de las unidades a nuestro repositorio de github en la rama principal main:

This screenshot shows a GitHub repository page for 'KarlaDenisseCruzSolis/ProyectosCompartidos'. The repository has 1 watching and 0 forks. It contains 19 projects listed in a table, mostly related to Kotlin and Android development. The repository has 0 releases, 0 packages published, and 54.5% Kotlin and 45.5% Java in its tech stack. Suggested workflows include 'Publish Java Package with Gradle' and 'Clojure'.

Project	Description	Last Updated
20_EjemplosApp	Subida de códigos de las lecciones de playground	5 days ago
Aprendizaje_Kotlin	Agregando AK cap 34 al 38	3 weeks ago
GreetingCard2	Agregando dessert_release	3 weeks ago
HappyBirthday	Fusionando diseño con los cambios de Yahir HP	2 months ago
Kotlin_Ya	Agregando Capítulo 8	last month
Lecciones_Playground_Kotlin	Agregando archivo U3_Parametros playground	3 days ago
TarjetaPresentacion	08/05/25 Proyecto Tarjeta de Presentación	5 days ago
android-basics-kotlin-sql-basics-app	Añadir proyecto SQL ruta 1 al repositorio corregido final	last week
art-space-app-master	09/05/25 10:56 Proyecto Art Space	4 days ago
basic-android-kotlin-compose-training-affirm...	Subida de archivos corregidos juice y etc	2 months ago
basic-android-kotlin-compose-training-amphi...	Nuevas carpetas agregadas	2 months ago
basic-android-kotlin-compose-training-bus-sc...	Agregados comentarios explicativos a bus schedule	last week
basic-android-kotlin-compose-training-courses	Agregando proyecto courses corregido final 2	last week
basic-android-kotlin-compose-training-cupcake	Agregando correctamente Cupcake, Dessert Clicker y Unscr...	2 months ago
basic-android-kotlin-compose-training-desser...	Agregando correctamente Cupcake, Dessert Clicker y Unscr...	2 months ago
basic-android-kotlin-compose-training-invent...	Agregando correctamente Cupcake, Dessert Clicker y Unscr...	2 months ago
basic-android-kotlin-compose-training-juice-t...	Subida de archivos corregidos juice y etc	2 months ago
basic-android-kotlin-compose-training-lemon...	Subida de archivos corregidos juice y etc	2 months ago
basic-android-kotlin-compose-training-mars-...	agregando carpeta de fotos	2 months ago
basic-android-kotlin-compose-training-practic...	Problemas prácticos: Conceptos básicos de Compose	last week
basic-android-kotlin-compose-training-race-tr...	Subida de archivos corregidos juice y etc	2 months ago
basic-android-kotlin-compose-training-reply-a...	Agregando Reply app al repositorio	2 months ago
basic-android-kotlin-compose-training-tip-cal...	Subida de archivos corregidos juice y etc	2 months ago
basic-android-kotlin-compose-training-unscr...	Agregando correctamente Cupcake, Dessert Clicker y Unscr...	2 months ago
basic-android-kotlin-compose-training-woof	Agregando calculator y demás	2 months ago
flight-search-app-android-kotlin-compose	Agregando comentarios a la app flight de vuelos	last week
lunch-tray6	Subiendo proyecto corregido lunch	2 months ago

This screenshot shows a GitHub repository page for 'KarlaDenisseCruzSolis/ProyectosCompartidos'. The repository has 1 watching and 0 forks. It contains 19 projects listed in a table, mostly related to Kotlin and Android development. The repository has 0 releases, 0 packages published, and 54.5% Kotlin and 45.5% Java in its tech stack. Suggested workflows include 'Publish Java Package with Gradle' and 'Clojure'.

Project	Description	Last Updated
basic-android-kotlin-compose-training-desser...	Agregando correctamente Cupcake, Dessert Clicker y Unscr...	2 months ago
basic-android-kotlin-compose-training-desser...	Agregados comentarios explicativos a dessert release	last week
basic-android-kotlin-compose-training-dice-r...	Subida de archivos corregidos juice y etc	2 months ago
basic-android-kotlin-compose-training-invent...	Agregados comentarios explicativos a inventory app 3	last week
basic-android-kotlin-compose-training-juice-t...	Subida de archivos corregidos juice y etc	2 months ago
basic-android-kotlin-compose-training-lemon...	Subida de archivos corregidos juice y etc	2 months ago
basic-android-kotlin-compose-training-mars-...	agregando carpeta de fotos	2 months ago
basic-android-kotlin-compose-training-practic...	Problemas prácticos: Conceptos básicos de Compose	last week
basic-android-kotlin-compose-training-race-tr...	Subida de archivos corregidos juice y etc	2 months ago
basic-android-kotlin-compose-training-reply-a...	Agregando Reply app al repositorio	2 months ago
basic-android-kotlin-compose-training-tip-cal...	Subida de archivos corregidos juice y etc	2 months ago
basic-android-kotlin-compose-training-unscr...	Agregando correctamente Cupcake, Dessert Clicker y Unscr...	2 months ago
basic-android-kotlin-compose-training-woof	Agregando calculator y demás	2 months ago
flight-search-app-android-kotlin-compose	Agregando comentarios a la app flight de vuelos	last week
lunch-tray6	Subiendo proyecto corregido lunch	2 months ago

The screenshot shows a GitHub repository page with a list of commits. The commits are as follows:

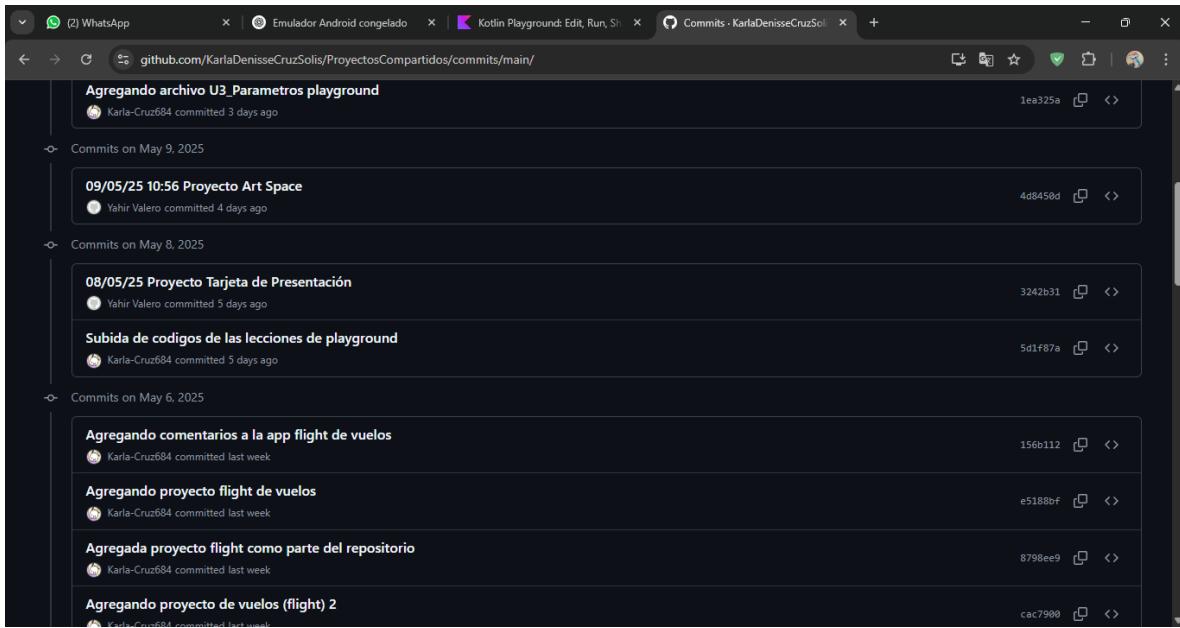
Commit	Description	Date
basic-android-kotlin-compose-training-juice-t...	Subida de archivos corregidos juice y etc	2 months ago
basic-android-kotlin-compose-training-lemon...	Subida de archivos corregidos juice y etc	2 months ago
basic-android-kotlin-compose-training-mars-...	agregando carpeta de fotos	2 months ago
basic-android-kotlin-compose-training-practic...	Problemas prácticos: Conceptos básicos de Compose	last week
basic-android-kotlin-compose-training-race-tr...	Subida de archivos corregidos juice y etc	2 months ago
basic-android-kotlin-compose-training-reply-a...	Agregando Reply app al repositorio	2 months ago
basic-android-kotlin-compose-training-tip-cal...	Subida de archivos corregidos juice y etc	2 months ago
basic-android-kotlin-compose-training-unscre...	Agregando correctamente Cupcake, Dessert Clicker y Unscr...	2 months ago
basic-android-kotlin-compose-training-woof	Agregando calcular y demás	2 months ago
flight-search-app-android-kotlin-compose	Agregando comentarios a la app flight de vuelos	last week
lunch-tray6	Subiendo proyecto corregido lunch	2 months ago
sports8	carpeta agregada	2 months ago
.swp	Agregando correctamente carpeta de 20 ejemplos	2 months ago
be4dc9e	Eliminando lunch	2 months ago
git checkout 3ad2e02	Aplicaciones agregadas	2 months ago

Los demás proyectos se encuentran en sus respectivas carpetas como en el caso de los 20\_EjemplosApp, el libro de Aprendizaje\_Kotlin, el libro de Kotlin\_Ya, GeeksforGeeks y los códigos de las Lecciones\_Playground\_Kotlin, mientras que la documentación está en la carpeta Documentacion\_Programas.

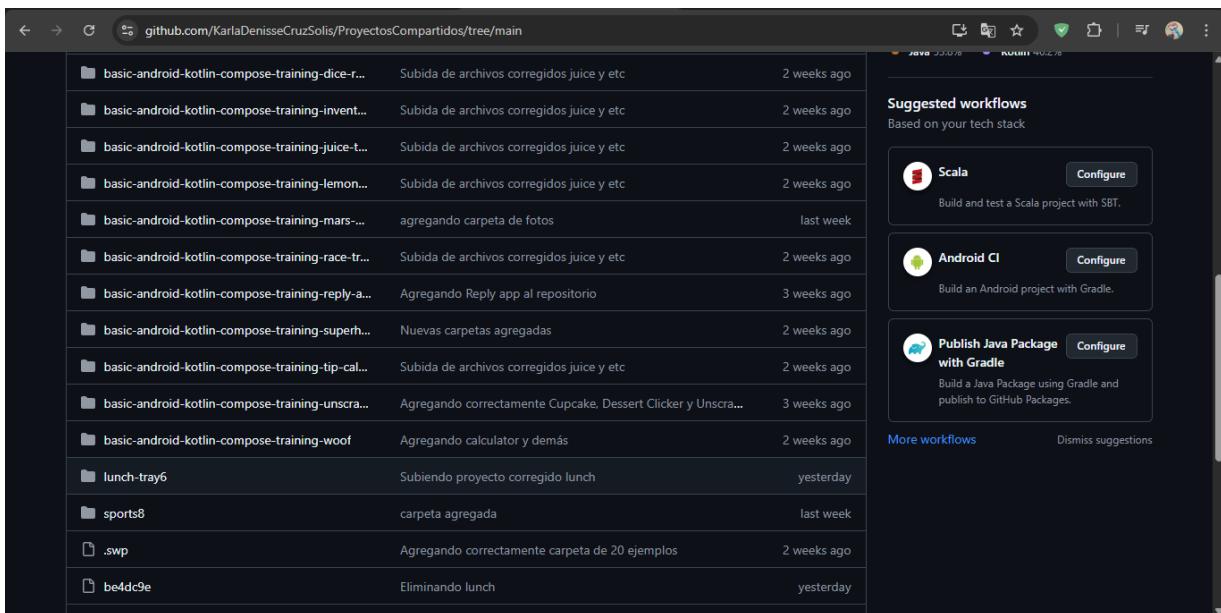
Además, también podemos ver el historial de algunas confirmaciones que hemos hecho:

The screenshot shows the GitHub 'Commits' page for the 'main' branch. The commits are grouped by date:

- Commits on Mar 21, 2025:
  - 21/03/25 Comentarios agregados en Capítulo 1 (Yahir Valero committed 7 hours ago, 485a409)
  - 21/03/25 Comentarios agregados en Capítulo 2 (Yahir Valero committed 8 hours ago, be4dc9e)
- Commits on Mar 20, 2025:
  - Cambiando diseño AK\_cap2\_diseño (Karla-Cruz684 committed yesterday, f83f3ef)
- Commits on Mar 19, 2025:
  - Diseño de AK\_HMundo\_diseño (Main y Comandos) (Karla-Cruz684 committed 2 days ago, 9482f98)



Conforme fuimos avanzando luego de un tiempo se corrigió la subida de algunos proyectos como el de lunch-tray6 y su contenido al repositorio; el problema era que se estaba inicializando como otro repositorio, así que lo corregimos con comandos como `git rm --cached -r lunch-tray6`, `rm -rf lunch-tray6/.git` y los volvimos a agregar con `git add ..`, `git commit -m "Subiendo proyecto corregido lunch"` y `git push origin main`:



## Gemini en Android Studio

Como se podía observar en las imágenes anteriores podemos ver que hay una carpeta llamada “20\_EjemplosApp” en nuestro repositorio, la cual corresponde a los

20 ejemplos que realizamos empleando Gemini en Android Studio. Esas versiones finales ya están y ya se habían hecho entrega en la tarea número 1.

Name	Last commit message	Last commit date
...		
BlocdeNotes	Agregando 8 aplicaciones (Yahir)	last week
CalculadoraMC	Agregando 8 aplicaciones (Yahir)	last week
Clima	Agregando 8 aplicaciones (Yahir)	last week
Contraseas	Agregando 8 aplicaciones (Yahir)	last week
ControldeGastos	Agregando 8 aplicaciones (Yahir)	last week
Ejemplo10	Añadiendo carpeta Aprendizaje_Kotlin	3 days ago
Ejemplo2	Ejemplo3 comentarios con main	last week
Ejemplo3	Comentarios de Ejemplo3	last week
Ejemplo4		
Ejemplo5		
Ejemplo6		

## Geeks for Geeks

Por otra parte, en nuestro repositorio en la carpeta GeeksforGeeks se encuentran algunas lecciones que realizamos en clase acerca del uso de los layouts que también ya se habían mostrado en clase.

Name	Last commit message	Last commit date
...		
ConstraintLayout28_03_25	Subiendo carpeta geeks	2 days ago
Linear_25_03_25	Subiendo carpeta geeks	2 days ago
ListV_java_03_04_25	Subiendo carpeta geeks	2 days ago
P_25_03_25	Subiendo carpeta geeks	2 days ago
Table_01_04_25	Subiendo carpeta geeks	2 days ago
WebService	Subiendo carpeta geeks	2 days ago
WebService2	Subiendo carpeta geeks	2 days ago
Web_V_02_04_25	Subiendo carpeta geeks	2 days ago
GreetingCard2		
HappyBirthday		

## Aprendizaje\_Kotlin

En cuanto al desarrollo de los ejercicios de Aprendizaje\_Kotlin, primeramente, abrimos el repositorio que ya tenemos sincronizado con git y git bash en Android Studio. En git bash nos colocamos en la carpeta correspondiente con el comando cd “C:\Users\Karla Cruz\AndroidStudioProjects\ProyectosCompartidos”.

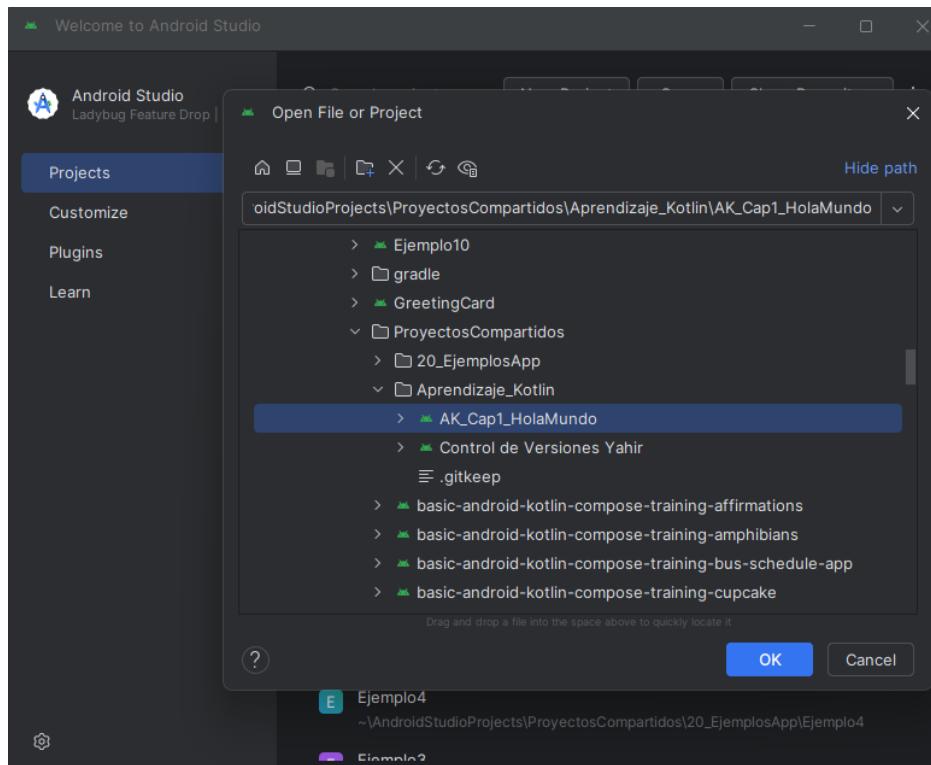
Posteriormente creamos la rama en la que vayamos a trabajar como se mencionaba anteriormente y nos colocamos en ella empezando a trabajar, haciendo las modificaciones necesarias.

Si estamos seguros de nuestros cambios hacemos un merge con la rama principal para que nuestro compañero pueda obtener o hacer un `git pull origin main` de esos cambios y seguir modificando. Posteriormente él también combina los cambios.

Podemos consultar las versiones con el comando `git log --oneline` y regresar a cualquiera de ellas o a la última que teníamos.

### Capítulo 1

Para el capítulo 1 del libro de Aprendizaje\_Kotlin se nos solicitaba un HolaMundo de distintas maneras, así como la lectura de entrada desde la línea de comandos.



En este caso se crearon las ramas AK\_HMundo\_diseño y AK\_HMundo\_comentarios, donde se pueden ver las versiones de la siguiente manera:

Modificación realizada	Versión
Proyecto recién subido	f07c28d
Diseño Karla sin comentarios de Yahir	9482f90
Proyecto actual de fusión del diseño Karla con el de Yahir	485a409 (main)

Como se puede observar, el proyecto recién subido se puede ver con git checkout f07c28d:

```
MINGW64:/c/Users/Karla Cruz/AndroidStudioProjects/ProyectosCompartid... — □ ×
Karla Cruz@JobitoPiece MINGW64 /
$ cd "C:\Users\Karla Cruz\AndroidStudioProjects\ProyectosCompartidos\Aprendizaje_Kotlin\AK_Cap1_HolaMundo"
```

```
Karla Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos/Aprendizaje_Kotlin/AK_Cap1_HolaMundo (main)
$ git log --oneline
9482f90 (HEAD -> main, origin/main, origin/HEAD, origin/AK_HMundo_diseño, AK_HMundo_diseño) Diseño de AK_HMundo_diseño (Main y Comandos)
39ef7db Merge branch 'main' of https://github.com/KarlaDenisseCruzSolis/ProyectosCompartidos
f07c28d Añadiendo AK_Cap1_HolaMundo configurado
```

```
MINGW64:/c/Users/Karla Cruz/AndroidStudioProjects/ProyectosCompartid... — □ ×
Karla Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos/Aprendizaje_Kotlin/AK_Cap1_HolaMundo (main)
$ git checkout f07c28d
M     Aprendizaje_Kotlin/AK_Cap1_HolaMundo/.idea/misc.xml
M     GreetingCard2/.idea/misc.xml
Note: switching to 'f07c28d'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at f07c28d Añadiendo AK_Cap1_HolaMundo configurado
```

The screenshot shows the Android Studio interface with the project 'AK\_Cap1\_HolaMundo' open. The left sidebar displays the project structure, including the 'src' directory which contains a 'my.program' package with files like Comandos.kt, Main.kt, Main2.kt, Main3.kt, and Vara.kt. The right panel shows the code editor with the 'Main.kt' file open. The code is as follows:

```
//Hola Mundo
@file:JvmName( name= "MyApp" )

package my.program

fun main(args: Array<String>) {
    println("Hello, world!")
}
```

The screenshot shows the Android Studio interface with the project 'AK\_Cap1\_HolaMundo' open. The left sidebar displays the project structure, including the 'src' directory which contains a 'my.program' package with files like Comandos.kt, Main.kt, Main2.kt, Main3.kt, and Vara.kt. The right panel shows the code editor with the 'Main2.kt' file open. The code is as follows:

```
//Hola Mundo usando declaración de objeto
package my.program

object Main2 {
    @JvmStatic
    fun main(args: Array<String>) {
        println("Hello World")
    }
}
```

A screenshot tool overlay is visible in the bottom right corner, indicating that a screenshot has been taken and saved to the clipboard.

The screenshot shows the Android Studio interface with the project 'AK\_Cap1\_HolaMundo' open. The code editor displays 'Main3.kt' with the following content:

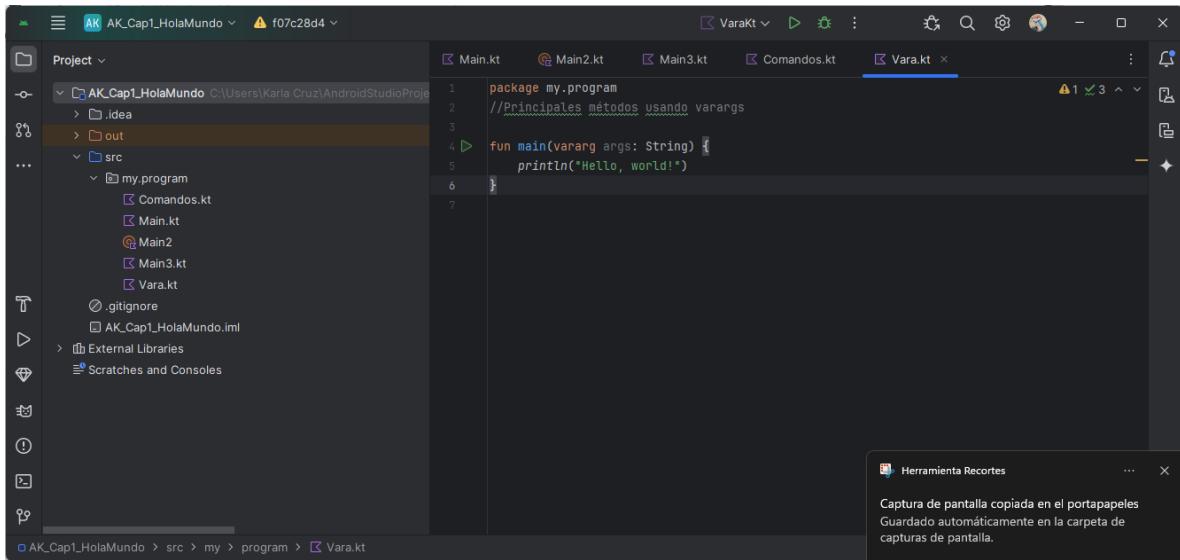
```
1 //Hola Mundo usando un objeto compañero
2 package my.program
3
4 class App {
5     companion object {
6         @JvmStatic
7         fun main(args: Array<String>) {
8             println("Hello World")
9         }
10    }
11 }
```

The status bar at the bottom indicates the file path: AK\_Cap1\_HolaMundo > src > my > program > Main3.kt.

The screenshot shows the Android Studio interface with the project 'AK\_Cap1\_HolaMundo' open. The code editor displays 'Comandos.kt' with the following content:

```
1 //Lectura de entrada desde la línea de comandos
2 fun main(args: Array<String>) {
3     println("Enter Two numbers")
4     var (a, b) = readLine()!!.split( ...delimiters: ' ' ) // !! operator to avoid NullPointerException
5     println("Max number is : ${maxNum(a.toInt(), b.toInt())}")
6
7
8
9
10
11
12
13
14
15
16
17
18 }
```

The status bar at the bottom indicates the file path: AK\_Cap1\_HolaMundo > src > my > program > Comandos.kt > maxNum.



Por otra parte, si queremos volver a la última versión (que en este caso es la del diseño, ya que aún no se le agregan los comentarios) podemos hacerlo mediante `git checkout 9482f90` o `git checkout main`:

```

MINGW64:/c/Users/Karla Cruz/AndroidStudioProjects/ProyectosCompartidos/Aprendizaje_Kotlin/AK_Cap1_HolaMundo (main)
$ git checkout 9482f90
M     Aprendizaje_Kotlin/AK_Cap1_HolaMundo/.idea/misc.xml
M     GreetingCard2/.idea/misc.xml
Note: switching to '9482f90'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 9482f90 Diseño de AK_HMundo_diseño (Main y Comandos)

```

```

Karla Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos/Aprendizaje_Kotlin/AK_Cap1_HolaMundo ((9482f90...))
$ git checkout main
M     Aprendizaje_Kotlin/AK_Cap1_HolaMundo/.idea/misc.xml
M     GreetingCard2/.idea/misc.xml
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

```

Esto nos devuelve a la versión del diseño (o la última versión (main) por el momento), en la cual modificamos los archivos Main.kt, y Comandos.kt:

The screenshot shows the Android Studio interface with the project 'AK\_Cap1\_HolaMundo' open. The code editor displays the 'Main.kt' file:

```
//Hola Mundo
@file:JvmName( name= "MyApp" )
package my.program

fun main(args: Array<String>) {
    println("¡Bienvenidos, estoy modificando Hola Mundo!") // Cambié el mensaje original por
}
```

The run tab shows the output of the program:

```
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:C:\Program Files\Android\Android Studio\lib\idea_rt.jar=64723:C:\Program Files\Android\Android Studio\lib" ¡Bienvenidos, estoy modificando Hola Mundo!
Process finished with exit code 0
```

The screenshot shows the Android Studio interface with the project 'AK\_Cap1\_HolaMundo' open. The code editor displays the 'Main2.kt' file:

```
//Hola Mundo usando declaración de objeto
package my.program

object Main2 {
    @JvmStatic
    fun main(args: Array<String>) {
        println("Hello World")
    }
}
```

The run tab shows the output of the program:

```
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:C:\Program Files\Android\Android Studio\lib\idea_rt.jar=64725:C:\Program Files\Android\Android Studio\lib" Hello World
Process finished with exit code 0
```

The screenshot shows the Android Studio interface with the project 'AK\_Cap1\_HolaMundo' open. The code editor displays Main3.kt with the following content:

```
//Hola Mundo usando un objeto compañero
package my.program

class App {
    companion object {
        @JvmStatic
        fun main(args: Array<String>) {
            println("Hello World")
        }
    }
}
```

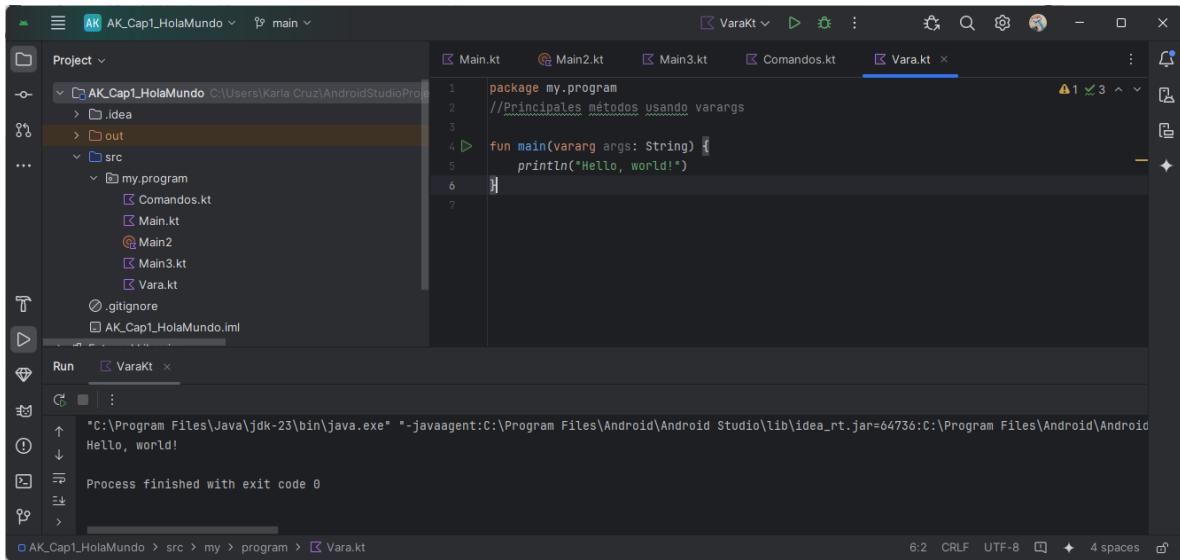
The run log shows the output: "Hello World" and "Process finished with exit code 0".

The screenshot shows the Android Studio interface with the project 'AK\_Cap1\_HolaMundo' open. The code editor displays ComandosKt with the following content:

```
//Lectura de entrada desde la línea de comandos
fun main(args: Array<String>) {
    println("Ingresá dos números separados por espacio:") //Cambié mensaje original
    var input: String?
    var validInput = false
    var a: Int
    var b: Int

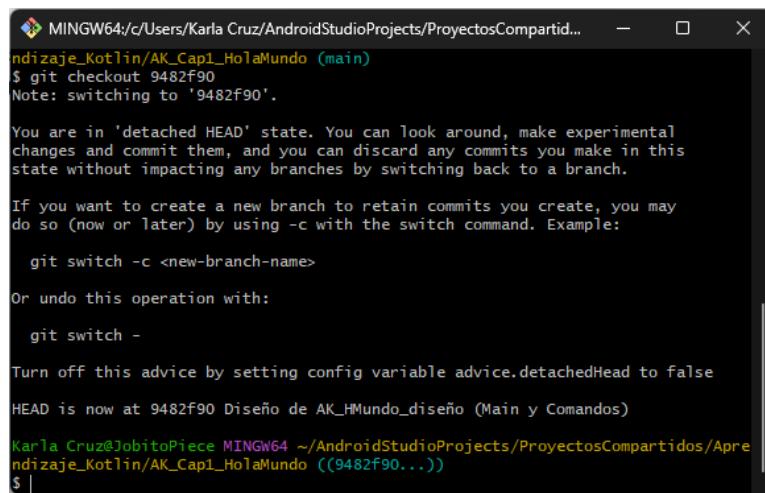
    // Bucle para asegurar que los datos sean números enteros
    while (!validInput) {
        input = readLine()
        // Intentar leer los números y validarlos
        if (input != null && input.split(" ").size == 2) {
```

The run log shows the interaction: "Ingresá dos números separados por espacio:", "10 2", "El valor de a es 10", and "El número máximo es: 10".



```
package my.program
//Principales métodos usando varargs
fun main(vararg args: String) {
    println("Hello, world!")
}
```

Al ya haberse agregado los comentarios, la versión que teníamos del diseño, donde se le agregó una validación de que fueran números enteros; esta pasa a ser [git checkout 9482f90](#), y la fusión de los comentarios pasa a ser la main.



```
MINGW64:/c/Users/Karla Cruz/AndroidStudioProjects/ProyectosCompartidos... - X
ndizaje_Kotlin/AK_Cap1_HolaMundo (main)
$ git checkout 9482f90
Note: switching to '9482f90'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

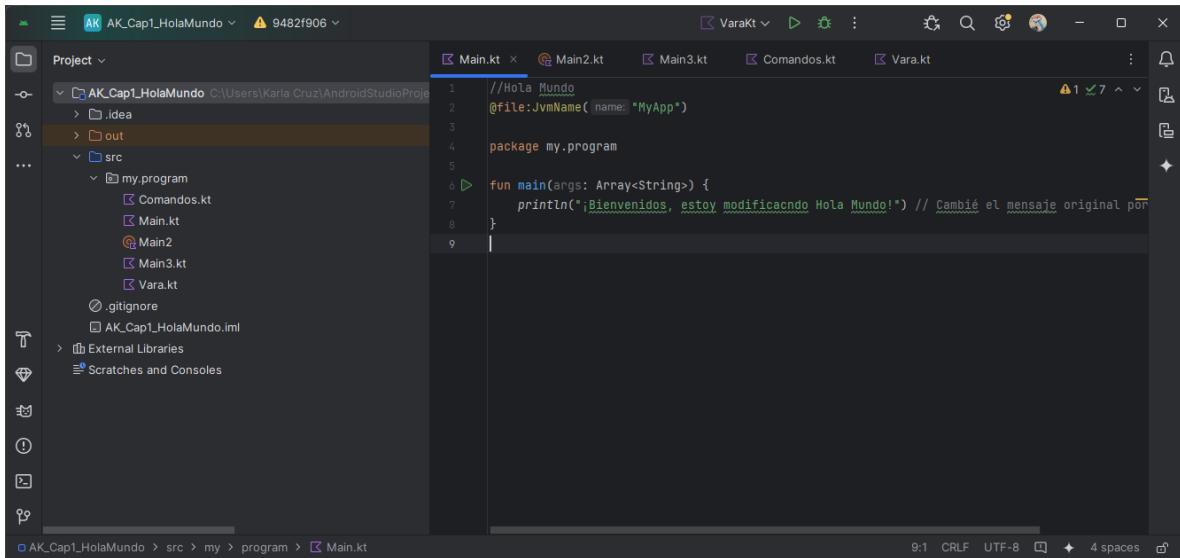
Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 9482f90 Diseño de AK_HMundo_diseño (Main y Comandos)

Karla Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos/Apre
ndizaje_Kotlin/AK_Cap1_HolaMundo ((9482f90...))
$ |
```

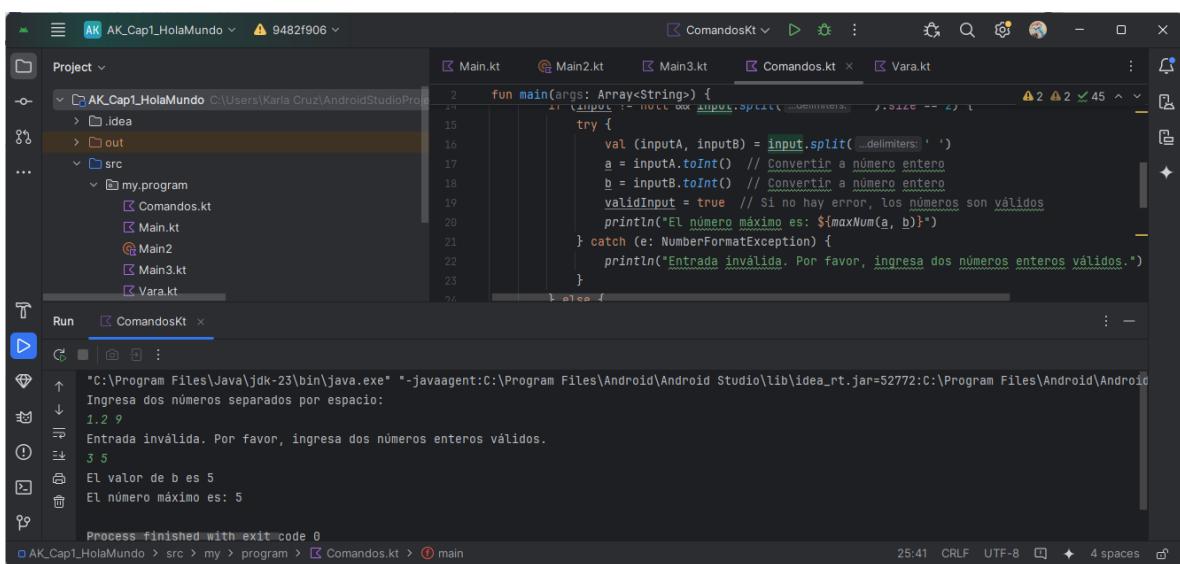


The screenshot shows the Android Studio interface with the project 'AK\_Cap1\_HolaMundo'. The 'Main.kt' file is open in the editor. The code prints 'Hello World' and then changes the message to 'Bienvenidos, estoy modificando Hola Mundo!'.

```
//Hola Mundo
@file:JvmName( name = "MyApp" )

package my.program

fun main(args: Array<String>) {
    println("Bienvenidos, estoy modificando Hola Mundo!") // Cambié el mensaje original por
}
```



The screenshot shows the Android Studio interface with the project 'AK\_Cap1\_HolaMundo'. The 'ComandosKt.kt' file is open in the editor. The code reads two integers from the user, finds the maximum, and prints it.

```
fun main(args: Array<String>) {
    val input = readLine() // Leer los dos números separados por espacio
    val inputA = input.split(' ').size == 2 ? input[0] : null
    val inputB = input[1]
    try {
        val inputA = inputA.toInt() // Convertir a número entero
        val inputB = inputB.toInt() // Convertir a número entero
        validInput = true // Si no hay error, los números son válidos
        println("El número máximo es: ${maxNum(a, b)}")
    } catch (e: NumberFormatException) {
        println("Entrada inválida. Por favor, ingresa dos números enteros válidos.")
    }
}
```

The 'Run' tab shows the output of the program:

```
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:C:\Program Files\Android\Android Studio\lib\idea_rt.jar=52772:C:\Program Files\Android\Android Studio\lib\proguard.jar" -Dfile.encoding=UTF-8 AK_Cap1_HolaMundo
Ingresá dos números separados por espacio:
1 2 9
Entrada inválida. Por favor, ingresa dos números enteros válidos.
3 5
El valor de a es 5
El número máximo es: 5
```

Y ya nuestra main fusionada con los comentarios:

```
Karla Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos/Aprendizaje_Kotlin/AK_Cap1_HolaMundo (main)
$ git checkout main
Already on 'main'
Your branch is up to date with 'origin/main'.
```

```

1 // Define el nombre del archivo generado en Java como "MyApp"
2 @fileJvmName("MyApp")
3 // Define el paquete al que pertenece este archivo
4 package my.program
5 // Función principal que se ejecuta al iniciar el programa
6 fun main(args: Array<String>) {
7     println("Bienvenidos, estoy modificando Hola Mundo!") // Imprime un mensaje de bienvenida
8 }

```

```

2 fun main(args: Array<String>) {
3     var a: Int // Variable para el primer número
4     var b: Int // Variable para el segundo número
5
6     // Bucle que se repite hasta que el usuario ingrese una entrada válida
7     while (!validInput) {
8         input = readLine() // Leer la entrada del usuario
9
10        // Verificar si la entrada es válida (no nula y contiene exactamente dos valores)
11        if (input != null && input.split(' ').size == 2) {
12            try { // Divide la entrada en dos valores y los convierte a enteros
13                val (inputA, inputB) = input.split(' ')
14                a = inputA.toInt() // Convertir el primer número a entero
15                b = inputB.toInt() // Convertir el segundo número a entero
16                validInput = true // Marcar la entrada como válida
17                println("El número máximo es: ${maxNum(a, b)}") // Llamar a la función maxNum
18            } catch (e: NumberFormatException) {
19                println("Entrada inválida. Por favor, ingresa exactamente dos números separados por un espacio")
20            }
21        } else {
22            println("Entrada inválida. Por favor, ingresa exactamente dos números separados por un espacio")
23        }
24    }
25 }

```

## Capítulo 2

Para el capítulo 2 del libro de Aprendizaje\_Kotlin se nos solicitaba llamar a un `toString()` en un tipo anulable, para lo cual tenemos las siguientes versiones:

Modificación realizada	Versión
Proyecto recién subido	8fbcd88
Diseño Karla sin comentarios de Yahir	f83f3ef
Proyecto actual de fusión del diseño Karla con el de los comentarios de Yahir	be4dc9e (main)

Creamos la rama de diseño y realizamos cambios para después subirlos a la rama principal:

```

Karla.Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos/Aprendizaje_Kotlin/Control de Versiones Yahir (main)
$ git checkout -b AK_cap2_diseño
Switched to a new branch 'AK_cap2_diseño'

Karla.Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos/Aprendizaje_Kotlin/Control de Versiones Yahir (AK_cap2_diseño)
$ git push origin AK_cap2_diseño
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'AK_cap2_diseño' on GitHub by visiting:
remote:     https://github.com/KarlaDenisseCruzSolis/ProyectosCompartidos/pull/
new/AK_cap2_dise%C3%B1o
remote:
To https://github.com/KarlaDenisseCruzSolis/ProyectosCompartidos.git
 * [new branch]      AK_cap2_diseño -> AK_cap2_diseño

```

Main.kt

```

1 fun main() {
2     Capitulo2()
3     //Capitulo3()
4     //Capitulo4()
5     //Capitulo5()
6 }

```

Run

"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:C:\Program Files\Android\Android Studio\lib\idea\_rt.jar=65420:C:\Program Files\Android\Android Studio\lib\javafx\_rt.jar=65421" -Dfile.encoding=UTF-8

Capítulo 2: Tipo anulable

Incorrecto (textField1): Hola, cambiando mensaje de Kotlin

Incorrecto (textField2):

Correcto (textField1): Hola, cambiando mensaje de Kotlin

Correcto (textField2):

Control de Versiones Yahir > src > Main.kt > main

2:5 CRLF UTF-8 4 spaces

Capítulo2.kt

```

1 fun Capitulo2() {
2     println("Capítulo 2: $incorrectText1\n")
3     // Simulación de un campo de texto que puede ser null
4     val textField1: String? = "Hola, cambiando mensaje de Kotlin" // Campo con texto
5     val textField2: String? = null // Campo nulo
6
7     // Ejemplo INCORRECTO: llamar a toString() directamente sobre un tipo anulable
8     val incorrectText1 = textField1?.toString() ?: ""
9     val incorrectText2 = textField2?.toString() ?: ""
10
11    println("Incorrecto (textField1): $incorrectText1") // "Hola, Kotlin"
12    println("Incorrecto (textField2): $incorrectText2") // "null"
13
14    // Ejemplo CORRECTO: Usar toString() solo si el valor no es null
15    val correctText1 = textField1?.toString() ?: ""
16    val correctText2 = textField2?.toString() ?: ""
17
18    println("Correcto (textField1): $correctText1") // "Hola, Kotlin"
19    println("Correcto (textField2): $correctText2") // "" (Cadena vacía)
20    println("")
21 }

```

Control de Versiones Yahir > src > Capítulo2.kt > Capítulo2

19:59 CRLF UTF-8 4 spaces

Guardamos cambios y fusionamos con la rama main:

```
Karla.Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos/Aprendizaje_Kotlin/Control de Versiones Yahir (AK_cap2_diseño)
$ git add .

Karla.Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos/Aprendizaje_Kotlin/Control de Versiones Yahir (AK_cap2_diseño)
$ git commit -m "Cambiando diseño AK_cap2_diseño"
[AK_cap2_diseño f83f3ef] Cambiando diseño AK_cap2_diseño
 5 files changed, 627 insertions(+), 13 deletions(-)
  create mode 100644 Aprendizaje_Kotlin/Control de Versiones Yahir/.idea/caches/deviceStreaming.xml

Karla.Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos/Aprendizaje_Kotlin/Control de Versiones Yahir (AK_cap2_diseño)
$ git push origin AK_cap2_diseño
Enumerating objects: 23, done.
Counting objects: 100% (23/23), done.
Delta compression using up to 8 threads
Compressing objects: 100% (11/11), done.
Writing objects: 100% (13/13), 2.75 KiB | 1.38 MiB/s, done.
Total 13 (delta 8), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (8/8), completed with 8 local objects.
To https://github.com/KarlaDenisseCruzSolis/ProyectosCompartidos.git
```

```
Karla.Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos/Aprendizaje_Kotlin/Control de Versiones Yahir (AK_cap2_diseño)
$ git push origin AK_cap2_diseño
Enumerating objects: 23, done.
Counting objects: 100% (23/23), done.
Delta compression using up to 8 threads
Compressing objects: 100% (11/11), done.
Writing objects: 100% (13/13), 2.75 KiB | 1.38 MiB/s, done.
Total 13 (delta 8), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (8/8), completed with 8 local objects.
To https://github.com/KarlaDenisseCruzSolis/ProyectosCompartidos.git
 9482f90..f83f3ef AK_cap2_diseño -> AK_cap2_diseño

Karla.Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos/Aprendizaje_Kotlin/Control de Versiones Yahir (AK_cap2_diseño)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

Karla.Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos/Aprendizaje_Kotlin/Control de Versiones Yahir (main)
$ git merge AK_cap2_diseño
Updating 9482f90..f83f3ef
Fast-forward
```

```

Karla_Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos/Aprendizaje_Kotlin/Control de Versiones Yahir (main)
$ git merge AK_cap2_diseño
Updating 9482f90..f83f3ef
Fast-forward
  ./idea/caches/deviceStreaming.xml      | 607 ++++++
  ./idea/libraries/KotlinJavaRuntime.xml | 20 ++
  ./Control de Versiones Yahir/.idea/vcs.xml | 1 +
  ./Control de Versiones Yahir/src/Capitulo2.kt | 8 ++
  ./Control de Versiones Yahir/src/Main.kt | 4 ++
5 files changed, 627 insertions(+), 13 deletions(-)
create mode 100644 Aprendizaje_Kotlin/Control de Versiones Yahir/.idea/caches/deviceStreaming.xml

Karla_Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos/Aprendizaje_Kotlin/Control de Versiones Yahir (main)
$ git push origin main
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/KarlaDenisseCruzSolis/ProyectosCompartidos.git
  9482f90..f83f3ef  main -> main

Karla_Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos/Aprendizaje_Kotlin/Control de Versiones Yahir (main)

```

Como podemos observar ya se reflejan los cambios:

```

Karla_Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos/Aprendizaje_Kotlin/Control de Versiones Yahir (main)
$ git log --oneline
f83f3ef (HEAD -> main, origin/main, origin/HEAD, origin/AK_cap2_diseño, AK_cap2_diseño) Cambiando diseño AK_cap2_diseño

```

The screenshot shows a GitHub repository interface. On the left, there's a file tree for the 'main' branch. In the center, a specific file named 'Capítulo2.kt' is displayed. At the top of the code editor, it says 'Karla-Cruz684 Cambiando diseño AK\_cap2\_diseño' and 'f83f3ef · 4 minutes ago'. The code itself contains several println statements and variable declarations.

```

1 fun Capítulo2() {
2     println("Capítulo 2: Tipo anulado \n")
3     // Simulación de un campo de texto que puede ser null
4     val textField1: String? = "Hola, cambiando mensaje de Kotlin" // Campo con texto
5     val textField2: String? = null // Campo nulo
6
7     // Ejemplo INCORRECTO: Llamar a toString() directamente sobre un tipo anulado
8     val incorrectText1 = textField1?.toString() ?: ""
9     val incorrectText2 = textField2?.toString() ?: ""
10
11    println("Incorrecto (textField1): $incorrectText1" // "Hola, Kotlin"
12    println("Incorrecto (textField2): $incorrectText2" // "null"
13
14    // Ejemplo CORRECTO: Usar toString() solo si el valor no es null
15    val correctText1 = textField1?.toString() ?: ""
16    val correctText2 = textField2?.toString() ?: ""
17
18    println("Correcto (textField1): $correctText1" // "Hola, Kotlin"
19    println("Correcto (textField2): $correctText2" // "" (Cadena vacía)
20    println("")
21 }

```

Si quisiéramos regresar a la versión de cómo se había subido desde un principio, aún sin la modificación del diseño podemos ejecutar el comando de `git checkout 8fbcd88`:

```
fun Capítulo2() {
    println("Capítulo 2: Advertencias \n")
    // Simulación de un campo de texto que puede ser null
    val textField1: String? = "Hola, Kotlin" // Campo con texto
    val textField2: String? = null // Campo nulo

    // Ejemplo INCORRECTO: Llamar a toString() directamente sobre un tipo anulable
    val incorrectText1 = textField1?.toString() ?: ""
    val incorrectText2 = textField2?.toString() ?: ""

    println("Incorrecto (textField1): $incorrectText1" // "Hola, Kotlin"
    println("Incorrecto (textField2): $incorrectText2" // "null" ✗

    // Ejemplo CORRECTO: Usar toString() solo si el valor no es null
    val correctText1 = textField1?.toString() ?: ""
    val correctText2 = textField2?.toString() ?: ""

    println("Correcto (textField1): $correctText1" // "Hola, Kotlin"
    println("Correcto (textField2): $correctText2" // "" ✅ (Cadena vacía)
    println("")
}
```

```
fun main() {
    //Capítulo2()
    //Capítulo3()
    //Capítulo4()
    Capítulo5()
}
```

```
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:C:\Program Files\Android\Android Studio\lib\idea_rt.jar=64987:C:\Program Files\Android\Android Studio\lib" Capítulo 2: Advertencias
Capítulo 2: Advertencias
Incorrecto (textField1): Hola, Kotlin
Incorrecto (textField2):
Correcto (textField1): Hola, Kotlin
Correcto (textField2):
```

Si queremos volver de nuevo a la versión actual que teníamos que en este caso es la del diseño ya fusionado con el main sería `git checkout f83f3ef` o `git checkout main`, ya que esto se está haciendo antes de fusionar los comentarios:

```

MINGW64:/c/Users/Karla Cruz/AndroidStudioProjects/ProyectosCompartidos/Aprendizaje_Kotlin/Control de Versiones Yahir ((f8bcd88...))
$ git commit -m "Guardando cambios antes de cambiar de versión"
[detached HEAD cf7e618] Guardando cambios antes de cambiar de versión
2 files changed, 14 insertions(+), 7 deletions(-)

Karla Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos/Aprendizaje_Kotlin/Control de Versiones Yahir ((cf7e618...))
$ git checkout main
Warning: you are leaving 1 commit behind, not connected to
any of your branches:

      cf7e618 Guardando cambios antes de cambiar de versión

If you want to keep it by creating a new branch, this may be a good time
to do so with:

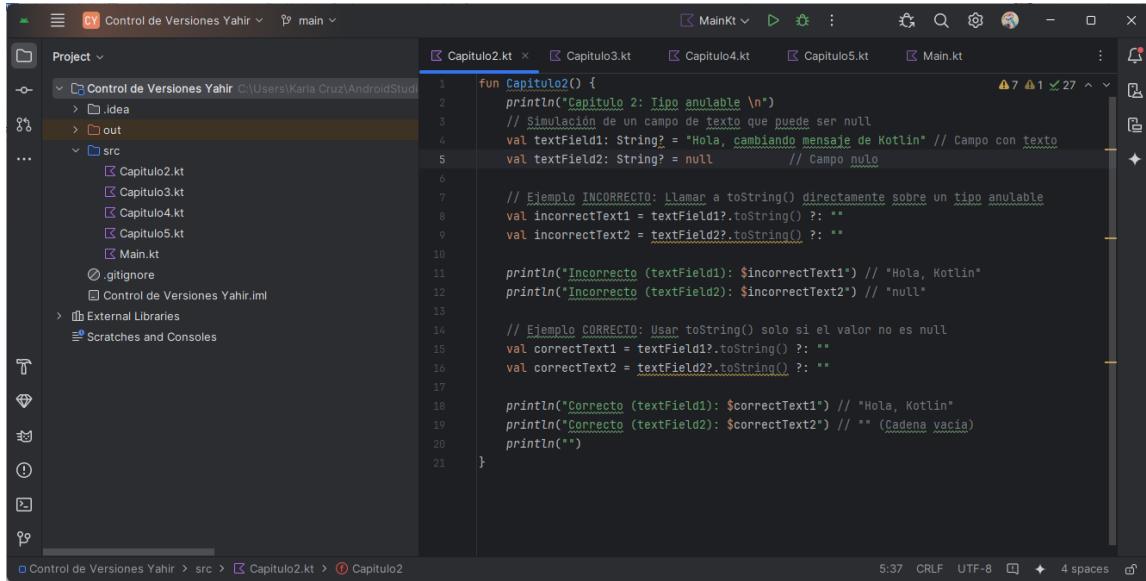
  git branch <new-branch-name> cf7e618

Switched to branch 'main'
Your branch is up to date with 'origin/main'.

Karla Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos/Aprendizaje_Kotlin/Control de Versiones Yahir (main)
$ |

```

Y volvimos:



The screenshot shows the Android Studio interface. On the left is the Project tool window, which lists the project structure: Control de Versiones Yahir (C:\Users\Karla Cruz\AndroidStudioProjects\ProyectosCompartidos\Aprendizaje\_Kotlin\Control de Versiones Yahir). Inside, there are folders .idea, out, and src, containing files like Capítulo2.kt, Capítulo3.kt, Capítulo4.kt, Capítulo5.kt, and Main.kt. The src folder is expanded. On the right is the Code Editor, showing the content of Capítulo2.kt. The code is as follows:

```

1 fun Capítulo2() {
2     println("Capítulo 2: Tipo anulado \n")
3     // Simulación de un campo de texto que puede ser null
4     val textField1: String? = "Hola, cambiando mensaje de Kotlin" // Campo con texto
5     val textField2: String? = null // Campo nulo
6
7     // Ejemplo INCORRECTO: Llamar a toString() directamente sobre un tipo anulado
8     val incorrectText1 = textField1?.toString() ?: ""
9     val incorrectText2 = textField2?.toString() ?: ""
10
11    println("Incorrecto (textField1): $incorrectText1") // "Hola, Kotlin"
12    println("Incorrecto (textField2): $incorrectText2") // "null"
13
14    // Ejemplo CORRECTO: Usar toString() solo si el valor no es null
15    val correctText1 = textField1?.toString() ?: ""
16    val correctText2 = textField2?.toString() ?: ""
17
18    println("Correcto (textField1): $correctText1") // "Hola, Kotlin"
19    println("Correcto (textField2): $correctText2") // "" (cadena vacía)
20    println("")
21 }

```

Ya con los comentarios fusionados quedaría la del diseño `git checkout f83f3ef` y la actual con los comentarios fusionados como `git checkout main`:

```
MINGW64:/c/Users/Karla Cruz/AndroidStudioProjects/ProyectosCompartidos/Control de Versiones Yahir (main)
$ git checkout f83f3ef
Note: switching to 'f83f3ef'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at f83f3ef Cambiando diseño AK_cap2_diseño

Karla Cruz@JobitoPiece MINGW64 ~/AndroidStudioProjects/ProyectosCompartidos/Control de Versiones Yahir ((f83f3ef...))
$
```

The screenshot shows the Android Studio interface. On the left, the Project tool window displays the project structure under 'Control de Versiones Yahir'. The 'src' folder contains files like Capítulo2.kt, Capítulo3.kt, Capítulo4.kt, Capítulo5.kt, and Main.kt. The 'out' folder is also visible. On the right, the main code editor window shows the content of Capítulo2.kt. The code demonstrates nullable string types and their handling:

```
1 fun Capítulo2() {
2     println("Capítulo 2: Tipo anulable \n")
3     // Simulación de un campo de texto que puede ser null
4     val textField1: String? = "Hola, cambiando mensaje de Kotlin" // Campo con texto
5     val textField2: String? = null // Campo nulo
6
7     // Ejemplo INCORRECTO: Llaman a toString() directamente sobre un tipo anulable
8     val incorrectText1 = textField1?.toString() ?: ""
9     val incorrectText2 = textField2?.toString() ?: ""
10
11    println("Incorrecto (textField1): $incorrectText1") // "Hola, Kotlin"
12    println("Incorrecto (textField2): $incorrectText2") // "null"
13
14    // Ejemplo CORRECTO: Usar toString() solo si el valor no es null
15    val correctText1 = textField1?.toString() ?: ""
16    val correctText2 = textField2?.toString() ?: ""
17
18    println("Correcto (textField1): $correctText1") // "Hola, Kotlin"
19    println("Correcto (textField2): $correctText2") // "" (Cadena vacía)
20    println("")
```

The screenshot shows the Android Studio interface with the project 'Control de Versiones Yahir' open. The Main.kt file is selected in the top navigation bar. The code in the editor is:

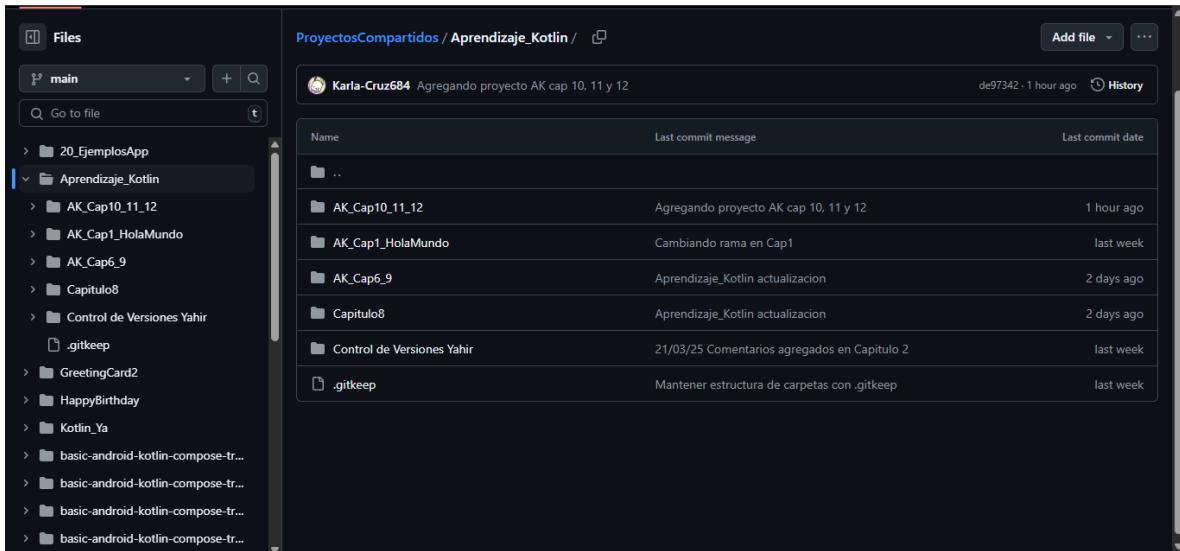
```
1 fun main() {
2     Capitulo02()
3     //Capitulo03()
4     //Capitulo04()
5     //Capitulo05()
6 }
```

Y ya agregando los comentarios esa versión se fusiona con nuestro main y queda de la siguiente manera:

The screenshot shows the Android Studio interface with the project 'Control de Versiones Yahir' open. The Main.kt file is selected in the top navigation bar. The code in the editor is:

```
1 fun Capitulo02() {
2     println("Capítulo 2: Tipo anulable \n") //Imprime el nombre de la actividad
3     // Declaración de variables anulables (pueden contener un valor o ser null)
4     val textField1: String? = "Hola, cambiando mensaje de Kotlin" // Variable con un valor definido
5     val textField2: String? = null // Variable con un valor null
6
7     // Ejemplo INCORRECTO: Llamar a toString() sobre un tipo anulable sin validación adecuada
8     val incorrectText1 = textField1?.toString() ?: "" // Si textField1 es null, devuelve ""
9     val incorrectText2 = textField2?.toString() ?: "" // Si textField2 es null, devuelve ""
10    // Imprimir los resultados incorrectos
11    println("Incorrecto ($textField1): $incorrectText1") // Muestra el valor de textField1 como correcto
12    println("Incorrecto ($textField2): $incorrectText2") // Muestra una cadena vacía en lugar de null
13
14    // Ejemplo CORRECTO: También usa el operador Elvis (?:) para manejar valores null de manera segura
15    val correctText1 = textField1?.toString() ?: "" // Devuelve el valor de textField1 o una cadena vacía si es null
16    val correctText2 = textField2?.toString() ?: "" // Devuelve una cadena vacía en caso de que textField2 sea null
17    // Imprimir los resultados correctos
18    println("Correcto ($textField1): $correctText1") // Muestra el valor de textField1
19    println("Correcto ($textField2): $correctText2") // Muestra "" (cadena vacía) en lugar de null
20
21 }
```

Cabe aclarar que en este proyecto tenemos del capítulo 2 al 5, pero posteriormente se agregaron los ejercicios del capítulo 6 al 9:



En el capítulo 6 se nos solicitaba crear una lista inmutable que contuviera tres elementos ("Item 1", "Item 2", "Item 3") para después imprimirlos en la consola:

```

1
2 fun main() {
3     // Crea una nueva lista de solo lectura que contiene elementos de tipo String
4     val list = listOf("Item 1", "Item 2", "Item 3")
5
6     // Imprime la lista en la consola. La salida será: "[Item 1, Item 2, Item 3]"
7     println(list)
8 }

```

También nos solicitaba crear un mapa inmutable donde las claves fueran enteros (1, 2, 3) y los valores fueran cadenas de texto ("Item 1", "Item 2", "Item 3") para después imprimirlo en la consola:

```

1 fun main() {
2     // Crea un nuevo mapa de solo lectura donde las claves son de tipo Integer y los valores son
3     val map = mapOf(Pair(1, "Item 1"), Pair(2, "Item 2"), Pair(3, "Item 3"))
4
5     // Imprime el mapa en la consola. La salida será: "{1=Item 1, 2=Item 2, 3=Item 3}"
6     println(map)
7 }
8
9

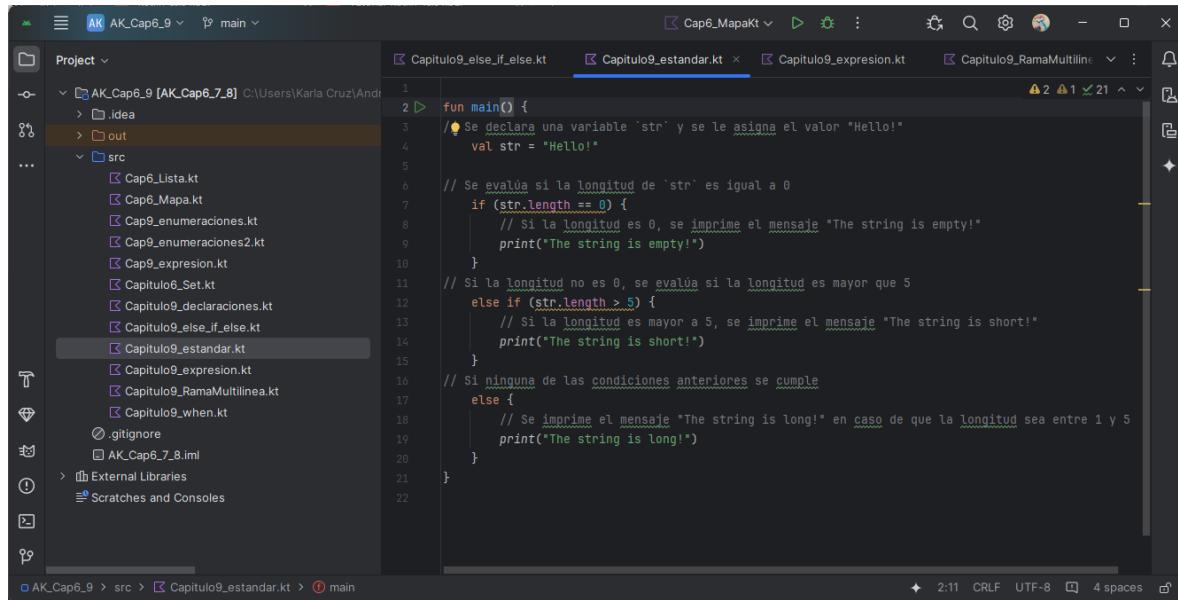
```

The screenshot shows the Android Studio interface with the project navigation bar on the left and the code editor on the right. The code editor displays a Kotlin file named Cap6\_Mapa.kt with the above code. The run log at the bottom shows the output of the program: "C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\Android\Android Studio\lib\idea\_rt.jar=61722:C:\Program Files\Android\Android Studio\lib\proguard.jar" -Dfile.encoding=UTF-8 Cap6\_MapaKt. The output in the log is: {1=Item 1, 2=Item 2, 3=Item 3}. Below the log, it says "Process finished with exit code 0".

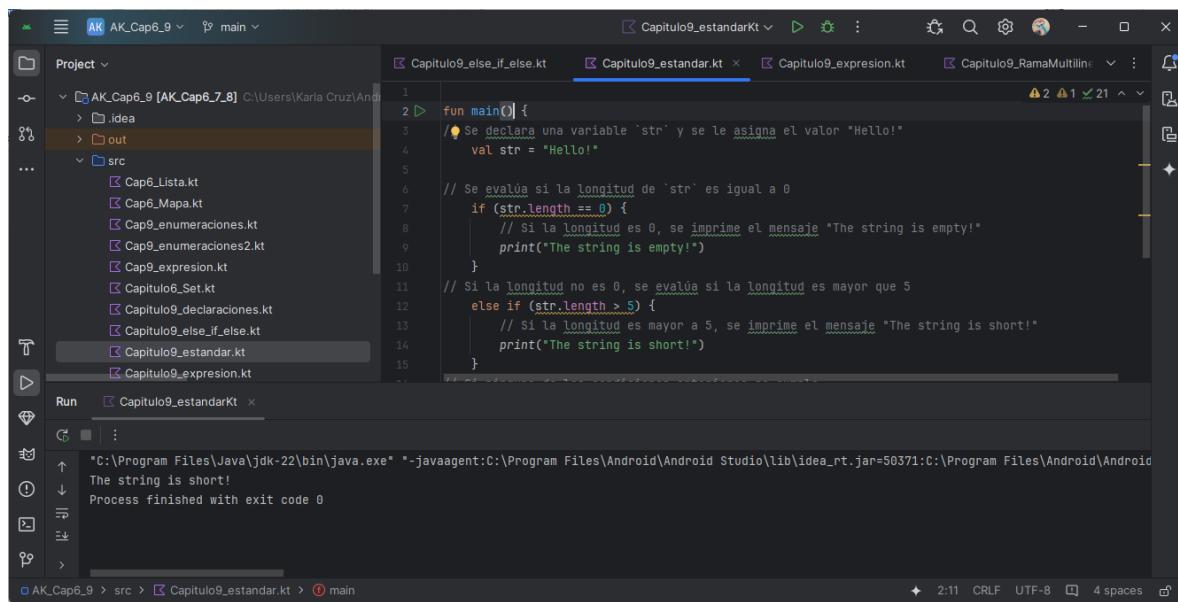
Por otra parte, en el capítulo 8 se pedía utilizar corrotinas en Kotlin para ejecutar código de manera asíncrona. El programa lanza una corrotina que imprime "World!" después de una pausa de 1 segundo, mientras el hilo principal imprime "Hello," de inmediato. Luego, el hilo principal espera 2 segundos para que la corrotina termine:

```

1 import kotlinx.coroutines.* // Necesario para GlobalScope, launch, delay y Dispatchers
2
3 fun main(args: Array<String>) {
4     // Inicia una nueva corrotina en el contexto predeterminado (usando Dispatchers.Default)
5     GlobalScope.launch(Dispatchers.Default) {
6         // Pausa la ejecución de la corrotina durante 1 segundo (sin bloquear el hilo principal)
7         delay( timeMillis: 1000L )
8
9         // Se imprime "World!" después de la pausa de 1 segundo
10        println("World!")
11    }
12
13    // Se imprime "Hello," inmediatamente en el hilo principal, mientras la corrotina está ejecutándose
14    println("Hello,")
15
16
17
18
19
20
21
22
23
24
25
26
27
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
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
305
306
307
308
309
309
310
311
312
313
313
314
315
316
316
317
318
319
319
320
321
322
323
323
324
325
326
326
327
328
329
329
330
331
332
332
333
334
335
335
336
337
337
338
339
339
340
341
341
342
343
343
344
345
345
346
347
347
348
349
349
350
351
351
352
353
353
354
355
355
356
357
357
358
359
359
360
361
361
362
363
363
364
365
365
366
367
367
368
369
369
370
371
371
372
373
373
374
375
375
376
377
377
378
379
379
380
381
381
382
383
383
384
385
385
386
387
387
388
389
389
390
391
391
392
393
393
394
395
395
396
397
397
398
399
399
400
401
401
402
403
403
404
405
405
406
407
407
408
409
409
410
411
411
412
413
413
414
415
415
416
417
417
418
419
419
420
421
421
422
423
423
424
425
425
426
427
427
428
429
429
430
431
431
432
433
433
434
435
435
436
437
437
438
439
439
440
441
441
442
443
443
444
445
445
446
447
447
448
449
449
450
451
451
452
453
453
454
455
455
456
457
457
458
459
459
460
461
461
462
463
463
464
465
465
466
467
467
468
469
469
470
471
471
472
473
473
474
475
475
476
477
477
478
479
479
480
481
481
482
483
483
484
485
485
486
487
487
488
489
489
490
491
491
492
493
493
494
495
495
496
497
497
498
499
499
500
501
501
502
503
503
504
505
505
506
507
507
508
509
509
510
511
511
512
513
513
514
515
515
516
517
517
518
519
519
520
521
521
522
523
523
524
525
525
526
527
527
528
529
529
530
531
531
532
533
533
534
535
535
536
537
537
538
539
539
540
541
541
542
543
543
544
545
545
546
547
547
548
549
549
550
551
551
552
553
553
554
555
555
556
557
557
558
559
559
560
561
561
562
563
563
564
565
565
566
567
567
568
569
569
570
571
571
572
573
573
574
575
575
576
577
577
578
579
579
580
581
581
582
583
583
584
585
585
586
587
587
588
589
589
590
591
591
592
593
593
594
595
595
596
597
597
598
599
599
600
601
601
602
603
603
604
605
605
606
607
607
608
609
609
610
611
611
612
613
613
614
615
615
616
617
617
618
619
619
620
621
621
622
623
623
624
625
625
626
627
627
628
629
629
630
631
631
632
633
633
634
635
635
636
637
637
638
639
639
640
641
641
642
643
643
644
645
645
646
647
647
648
649
649
650
651
651
652
653
653
654
655
655
656
657
657
658
659
659
660
661
661
662
663
663
664
665
665
666
667
667
668
669
669
670
671
671
672
673
673
674
675
675
676
677
677
678
679
679
680
681
681
682
683
683
684
685
685
686
687
687
688
689
689
690
691
691
692
693
693
694
695
695
696
697
697
698
699
699
700
701
701
702
703
703
704
705
705
706
707
707
708
709
709
710
711
711
712
713
713
714
715
715
716
717
717
718
719
719
720
721
721
722
723
723
724
725
725
726
727
727
728
729
729
730
731
731
732
733
733
734
735
735
736
737
737
738
739
739
740
741
741
742
743
743
744
745
745
746
747
747
748
749
749
750
751
751
752
753
753
754
755
755
756
757
757
758
759
759
760
761
761
762
763
763
764
765
765
766
767
767
768
769
769
770
771
771
772
773
773
774
775
775
776
777
777
778
779
779
780
781
781
782
783
783
784
785
785
786
787
787
788
789
789
790
791
791
792
793
793
794
795
795
796
797
797
798
799
799
800
801
801
802
803
803
804
805
805
806
807
807
808
809
809
810
811
811
812
813
813
814
815
815
816
817
817
818
819
819
820
821
821
822
823
823
824
825
825
826
827
827
828
829
829
830
831
831
832
833
833
834
835
835
836
837
837
838
839
839
840
841
841
842
843
843
844
845
845
846
847
847
848
849
849
850
851
851
852
853
853
854
855
855
856
857
857
858
859
859
860
861
861
862
863
863
864
865
865
866
867
867
868
869
869
870
871
871
872
873
873
874
875
875
876
877
877
878
879
879
880
881
881
882
883
883
884
885
885
886
887
887
888
889
889
890
891
891
892
893
893
894
895
895
896
897
897
898
899
899
900
901
901
902
903
903
904
905
905
906
907
907
908
909
909
910
911
911
912
913
913
914
915
915
916
917
917
918
919
919
920
921
921
922
923
923
924
925
925
926
927
927
928
929
929
930
931
931
932
933
933
934
935
935
936
937
937
938
939
939
940
941
941
942
943
943
944
945
945
946
947
947
948
949
949
950
951
951
952
953
953
954
955
955
956
957
957
958
959
959
960
961
961
962
963
963
964
965
965
966
967
967
968
969
969
970
971
971
972
973
973
974
975
975
976
977
977
978
979
979
980
981
981
982
983
983
984
985
985
986
987
987
988
989
989
990
991
991
992
993
993
994
995
995
996
997
997
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1
```



```
1 fun main() {
2     // Se declara una variable 'str' y se le asigna el valor "Hello!"
3     val str = "Hello!"
4
5     // Se evalúa si la longitud de 'str' es igual a 0
6     if (str.length == 0) {
7         // Si la longitud es 0, se imprime el mensaje "The string is empty!"
8         print("The string is empty!")
9     }
10    // Si la longitud no es 0, se evalúa si la longitud es mayor que 5
11    else if (str.length > 5) {
12        // Si la longitud es mayor a 5, se imprime el mensaje "The string is short!"
13        print("The string is short!")
14    }
15    // Si ninguna de las condiciones anteriores se cumple
16    else {
17        // Se imprime el mensaje "The string is long!" en caso de que la longitud sea entre 1 y 5
18        print("The string is long!")
19    }
20 }
```



```
Run Capítulo9_estandarKt
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\Android\Android Studio\lib\idea_rt.jar=50371:C:\Program Files\Android\Android Studio\lib\proguard.jar" -Dfile.encoding=UTF-8 Capítulo9_estandarKt
The string is short!
Process finished with exit code 0
```

Otro que se solicitó fue la declaración-if como una expresión. El código asigna un valor a str dependiendo de las condiciones evaluadas. Si if se usa como expresión, la rama else es obligatoria para garantizar que siempre haya un valor de retorno.

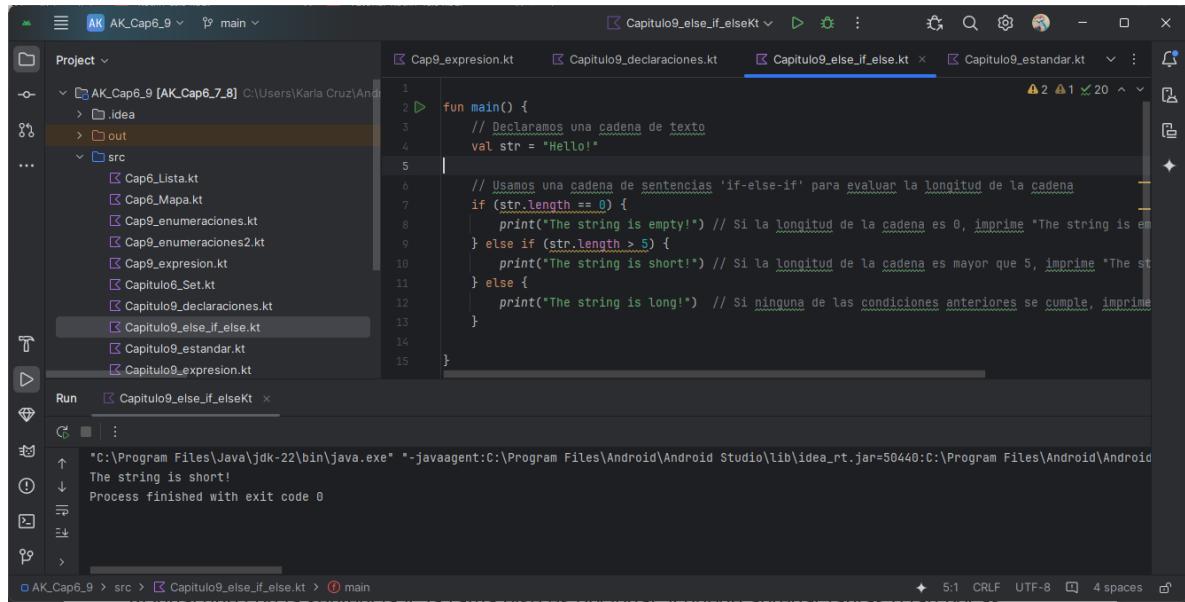
The screenshot shows the Android Studio interface with the project 'AK\_Cap6\_9' open. The code editor displays the file 'Capítulo9\_expresión.kt' containing the following Kotlin code:

```
1 fun main() {
2     // Definimos las condiciones
3     val condition1 = true // Por ejemplo, esta condición es verdadera
4     val condition2 = false // Esta condición es falsa
5
6     // Usamos la condición en una expresión 'if'
7     val str = if (condition1) {
8         "Condition1 met!" // Si 'condition1' es verdadera, asignamos "Condition1 met!" a 'str'.
9     } else if (condition2) {
10        "Condition2 met!" // Si 'condition1' es falsa pero 'condition2' es verdadera, asignamos "Condition2 met!" a 'str'.
11    } else {
12        "Conditions not met!" // Si ambas condiciones son falsas, asignamos "Conditions not met!" a 'str'.
13    }
14
15    // Imprime el valor de 'str'
16    println(str)
17
18 }
```

The screenshot shows the Android Studio interface with the project 'AK\_Cap6\_9' open. The code editor displays the file 'Capítulo9\_expresión.kt' with the same code as above. Below the editor, the 'Run' tool window is visible, showing the output of the run command:

```
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\Android\Android Studio\lib\idea_rt.jar=50428:C:\Program Files\Android\Android Studio\lib\proguard.jar" -Dfile.encoding=UTF-8 Capítulo9_expresiónKt
Condition1 met!
Process finished with exit code 0
```

Usando When-statement en lugar de if-else-if chains, el código usa when como alternativa a múltiples if-else if, evaluando condiciones secuencialmente. Si str está vacío, imprime "The string is empty!". Si tiene más de 5 caracteres, imprime "The string is short!". En cualquier otro caso, imprime "The string is long!". Funciona igual que el if-else if, pero con una sintaxis más clara y concisa.

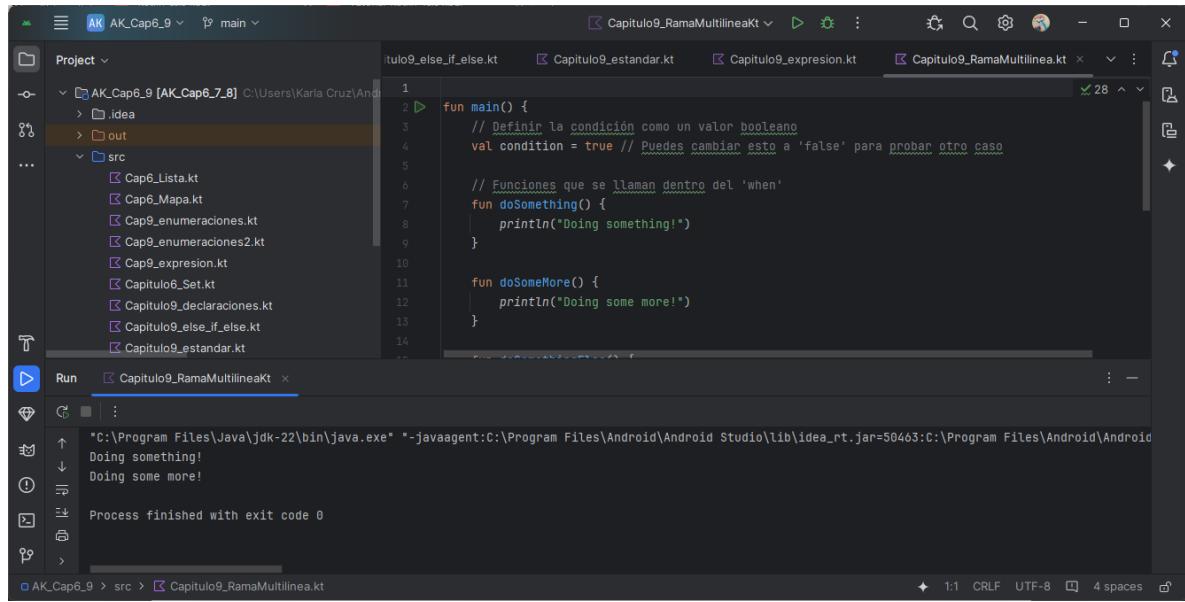


```
fun main() {
    // Declaramos una cadena de texto
    val str = "Hello!"

    // Usamos una cadena de sentencias 'if-else-if' para evaluar la longitud de la cadena
    if (str.length == 0) {
        println("The string is empty!") // Si la longitud de la cadena es 0, imprime "The string is empty!"
    } else if (str.length > 5) {
        println("The string is short!") // Si la longitud de la cadena es mayor que 5, imprime "The string is short!"
    } else {
        println("The string is long!") // Si ninguna de las condiciones anteriores se cumple, imprime "The string is long!"
    }
}
```

The string is short!  
Process finished with exit code 0

También se nos pidió un código que muestre que en una expresión when, la rama else es opcional y que se pueden agregar tantas condiciones como se deseé. Además, que se permita ejecutar múltiples instrucciones dentro de una rama usando llaves {}. Si condition es true, ejecuta doSomething() y doSomeMore(). Si ninguna condición se cumple, ejecuta doSomethingElse().



```
fun main() {
    // Definir la condición como un valor booleano
    val condition = true // Puedes cambiar esto a 'false' para probar otro caso

    // Funciones que se llaman dentro del 'when'
    fun doSomething() {
        println("Doing something!")
    }

    fun doSomeMore() {
        println("Doing some more!")
    }

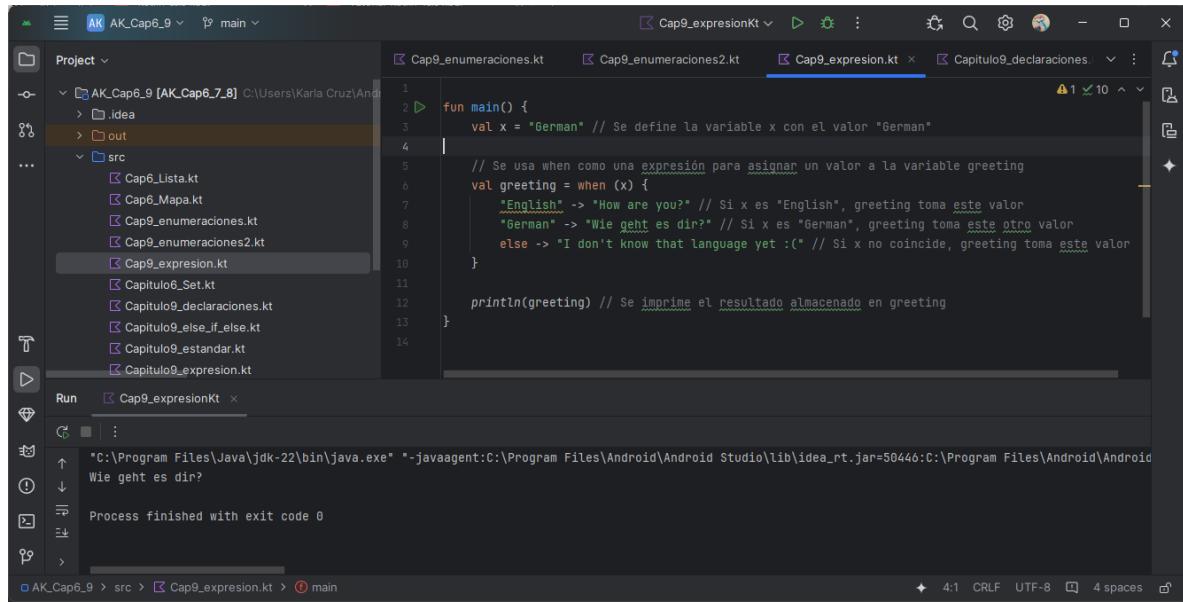
    fun doSomethingElse() {
        println("I don't know that language yet :(")
    }
}

When condition is true:
    doSomething()
    doSomeMore()

When condition is false:
    doSomethingElse()

Process finished with exit code 0
```

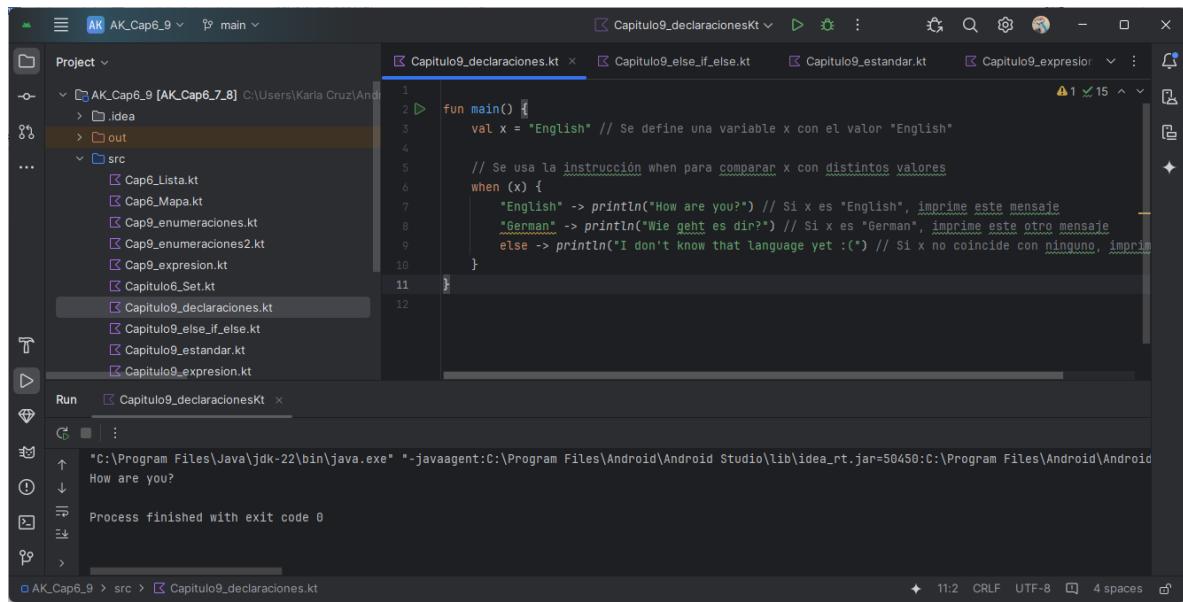
Por otra parte, cuando coinciden argumentos de declaración, el código usa when con un argumento (x), comparándolo secuencialmente con distintas opciones usando ==. Si x es "English", imprime "How are you?". Si es "German", imprime "Wie geht es dir?". Si no coincide con ninguna opción, ejecuta la rama else, imprimiendo "I don't know that language yet :(".



```
fun main() {
    val x = "German" // Se define la variable x con el valor "German"
    val greeting = when (x) {
        "English" -> "How are you?" // Si x es "English", greeting toma este valor
        "German" -> "Wie geht es dir?" // Si x es "German", greeting toma este otro valor
        else -> "I don't know that language yet :(" // Si x no coincide, greeting toma este valor
    }
    println(greeting) // Se imprime el resultado almacenado en greeting
}
```

"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\Android\Android Studio\lib\idea\_rt.jar=50446:C:\Program Files\Android\Android Studio\lib\proguard.jar" AK\_Cap6\_9 Cap9\_expresionKt  
Wie geht es dir?  
Process finished with exit code 0

Cuando se usa declaración como expresión el código usa when como una expresión, asignando el resultado a la variable greeting. Dependiendo del valor de x greeting tomará "How are you?", "Wie geht es dir?" o "I don't know that language yet :(". Luego, imprime greeting.



```
fun main() {
    val x = "English" // Se define una variable x con el valor "English"
    when (x) {
        "English" -> println("How are you?") // Si x es "English", imprime este mensaje
        "German" -> println("Wie geht es dir?") // Si x es "German", imprime este otro mensaje
        else -> println("I don't know that language yet :(") // Si x no coincide con ninguno, imprime este mensaje
    }
}
```

"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\Android\Android Studio\lib\idea\_rt.jar=50450:C:\Program Files\Android\Android Studio\lib\proguard.jar" AK\_Cap6\_9 Capitulo9\_declaracionesKt  
How are you?  
Process finished with exit code 0

Por último, cuando se usa declaración con enumeraciones, el código usa when para evaluar un valor de un enum (Day). Dependiendo del día, ejecutará diferentes acciones. Se pueden combinar múltiples valores en una misma rama (como Day.Monday, Day.Tuesday ->), lo que permite agrupar condiciones similares. Esto hace que when sea más limpio y eficiente al trabajar con enumeraciones.

```

1 // Se define una enumeración llamada Day con los días de la semana
2 enum class Day {
3     Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday
4 }
5
6 fun doOnDay(day: Day) {
7     // Se usa when para ejecutar diferentes acciones dependiendo del día
8     when (day) {
9         Day.Sunday -> println("Relax and enjoy!")
10        Day.Monday, Day.Tuesday -> println("Time to work!")
11        Day.Wednesday -> println("Halfway through the week!")
12        Day.Thursday -> println("Almost there!")
13        Day.Friday -> println("Weekend is coming!")
14        Day.Saturday -> println("Enjoy your weekend!")
15    }
16 }
17

```

Output:

```

"Halfway through the week!
Process finished with exit code 0

```

El siguiente código muestra que `when` debe ser exhaustivo al trabajar con enumeraciones. Si no se cubren todos los casos, el compilador generará un error. Para evitarlo, se puede agregar una rama `else`, que actúa como un caso predeterminado. Esto hace que `when` sea más seguro y natural que `if-else`, ya que garantiza que todos los valores del enum sean manejados.

```

1 // Definimos una enumeración llamada WeekDay que representa los días de la semana
2 enum class WeekDay {
3     Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday
4 }
5
6 // Definimos una función que toma un día de la semana como argumento y ejecuta diferentes acciones
7 fun doOnWeekDay(day: WeekDay) {
8     when (day) {
9         WeekDay.Monday -> println("Work")
10        WeekDay.Tuesday -> println("Work hard")
11        WeekDay.Wednesday -> println("Midweek tasks")
12        WeekDay.Thursday -> println("Almost Friday")
13        WeekDay.Friday -> println("Prepare for weekend")
14        else -> println("Party on weekend!")
15    }
16 }
17

```

Output:

```

"Party on weekend!
Process finished with exit code 0

```

Si hablamos acerca de los [capítulos 10, 11 y 12](#) empezamos con lo que se nos solicitó en el capítulo 10, el cual muestra la delegación de clases en Kotlin. En lugar de que Baz implemente la interfaz Foo directamente, delega su implementación a una instancia de Bar, que ya tiene la función `example()`, así cuando se llama a `Baz(Bar()).example()`, la ejecución se delega a `Bar.example()`, imprimiendo: Hello, world!

```

interface Foo {
    fun example() // Método a delegar
}

// Definir la clase Bar, que implementa el método example() de Foo
class Bar : Foo {
    override fun example() {
        println("Hello, world!") // Implementación del método
    }
}

// Definir la clase Baz, que delega la implementación de Foo a un objeto de tipo Bar
class Baz(b: Foo) : Foo by b // Delegación del comportamiento de la interfaz Foo a Bar

// Función main para ejecutar el ejemplo
fun main() {
}

```

El capítulo 11 el código muestra cómo usar funciones infijas para crear un DSL interno en Kotlin. La función shouldBe se utiliza para hacer comparaciones de manera más legible, permitiendo escribir expresiones como 100.plusOne() shouldBe 101 en lugar de una sintaxis más compleja, lo que mejora la claridad del código en pruebas.

```

package com.example.ak_cap11_12 // Define el paquete donde se encuentra la clase
import org.junit.Test // Importa la anotación @Test de JUnit para pruebas unitarias
import org.junit.Assert.assertEquals // Importa el método assertEquals para validaciones

// Función de extensión infix que compara si un valor es igual al esperado
infix fun <T> T?.shouldBe(expected: T?) = assertEquals(expected, actual)

```

Test Results 20ms

- Tests passed: 1 of 1 test – 20ms
- Task :app:bundleDebugClassesToCompileJar
- Task :app:processDebugJavaRes
- Task :app:compileDebugUnitTestKotlin
- Task :app:compileDebugUnitTestJavaWithJavac NO-SOURCE
- Task :app:processDebugUnitTestJavaRes
- Task :app:testDebugUnitTest
- Comparing 101 with 101
- BUILD SUCCESSFUL in 7s
- 22 actionable tasks: 7 executed, 15 up-to-date
- Build Analyzer results available
- 04:54:52 p. m.: Execution finished ':app:testDebugUnitTest --tests "com.example.ak\_cap11\_12.DSLTest.test"'.

También, se nos pide anular el método de invocación para construir DSL. El código muestra cómo usar la anulación del método invoke en Kotlin para crear un DSL y permite tratar una instancia de clase como si fuera una función, facilitando un estilo de código más expresivo y fluido, donde métodos como bigger se pueden llamar dentro de un bloque que usa el contexto de la instancia.

```

kt Cap11_DSLTest.kt Cap10_Metodo_Clase.kt Cap11_DSL_Invocacion.kt
10 }
11
12 fun main(args: Array<String>) {
13     // Creamos una instancia de MyExample con el valor 233
14     val ex = MyExample( 233 )
15
16     // Usamos el bloque ex { ... } para ejecutar código dentro de su contexto
17     ex {
18         // 'bigger' es accesible dentro del contexto de 'ex'
19         // Comprobamos si 777 es mayor que 233 y luego imprimimos un mensaje
20         if (777.bigger()) {
21             kotlin.io.println("why") // Esto imprimirá "why" porque 777 es mayor que 233
22         }
23     }
24 }

```

En cuanto a usar operadores con lambdas el código define un operador infijo personalizado rem que ejecuta una lambda si se cumple una condición aleatoria. En el ejemplo, 20 % { ... } tiene un 20% de posibilidad de imprimir un mensaje, creando un estilo de código más expresivo y similar a un DSL, pero en este caso se reemplazó por un 80%.

```

kt Cap10_Metodo_Clase.kt Cap11_DSL_Invocacion.kt Cap11_Operadores_lambdas.kt
4 val r = Random( seed: 233 )
5
6 // Función de extensión para el operador rem
7 infix inline operator fun Int.rem(block: () -> Unit) {
8     // Genera un número aleatorio entre 0 y 99
9     if (r.nextInt( until: 100 ) < this) {
10         block() // Ejecuta el bloque si la condición se cumple
11     }
12 }
13
14 // Ejemplo de uso
15 fun main() {
16     80 % { //cambie de 20 a 80
17         println("The possibility you see this message is 80%") //cambie de 20 a 80
18     }
19 }

```

Por último, en el capítulo 11 se nos solicita definir una extensión de operador para String que ejecuta un bloque de código, captura excepciones y las imprime en caso de error.

The screenshot shows the Android Studio interface with the project 'AK\_Cap10\_11\_12' open. The code editor displays a file named 'Cap11\_extensiones\_lambdas.kt'. The code defines a class 'AST' with a constructor that takes a string expression and returns the expression itself. It then defines an 'evaluate()' function that returns 2 if the expression is '1 + 1' and 0 otherwise. A main function is also present. The run tab shows the output of a test, indicating it started, executed a block, got a result of 2, compared it to 2, and finished with an exit code of 0.

```
class AST(val expression: String) {  
    // Lógica para construir el árbol de sintaxis (simplificado)  
    return this  
}  
  
fun evaluate(): Int {  
    // Evaluación de la expresión (simplificada para "1 + 1")  
    return if (expression == "1 + 1") 2 else 0  
}  
  
// Función principal donde se realiza la prueba  
fun main() {  
    // Verificar si todo es correcto  
}
```

En cuanto al capítulo 12 se muestra cómo las clases de enumeración en Kotlin pueden tener un constructor y ser inicializadas como cualquier otra clase. En este caso, la clase Color tiene un constructor que acepta un valor rgb (un entero representando el color). Las constantes de enumeración RED, GREEN y BLUE son inicializadas con sus respectivos valores rgb:

The screenshot shows the Android Studio interface with the project 'AK\_Cap10\_11\_12' open. The code editor displays a file named 'Cap12\_InicializacionKt.kt'. The code defines an enum class 'Color' with three values: RED, GREEN, and BLUE, each associated with a specific RGB value. The run tab shows the output of a test, listing the colors and their RGB values, then showing the result of a valueOf('GREEN') call, and finally a comparison between RED and BLUE.

```
// Definimos una enumeración de colores con un valor RGB asociado  
enum class Color(val rgb: Int) {  
    RED( rgb: 0xFFFF00),  
    GREEN( rgb: 0x00FF00),  
    BLUE( rgb: 0x0000FF)  
}
```

En las funciones y propiedades en enumeraciones el código en la clase Color tiene una propiedad abstracta `rgb`, que cada constante debe implementar, y una función `colorString()` que convierte el valor `rgb` a formato hexadecimal.

The screenshot shows the Android Studio interface with the project 'AK\_Cap10\_11\_12' open. The code editor displays a Kotlin file named 'Cap12\_Enumeracion.kt'. The code defines an enum class 'ColorWithFunctions' with three entries: RED, GREEN, and BLUE. Each entry has a custom implementation for the 'rgb' property. The terminal output shows the results of running the code, displaying the color names and their corresponding RGB values and hex codes.

```
package com.example.ak_cap11_12

// Definir la enumeración con funciones y propiedades
enum class ColorWithFunctions {
    RED {
        // Sobrescribimos la propiedad 'rgb' para el color rojo
        override val rgb: Int = 0xFFFF00 // Valor RGB de color rojo
    },
    GREEN {
        // Sobrescribimos la propiedad 'rgb' para el color verde
        override val rgb: Int = 0x00FF00 // Valor RGB de color verde
    },
    BLUE {
        // Sobrescribimos la propiedad 'rgb' para el color azul
        override val rgb: Int = 0x0000FF // Valor RGB de color azul
    }
}
```

```
*C:\Program Files\Android\Android Studio\jbr\bin\java.exe* ...
Lista de colores con funciones:
RED - RGB: 16711680 - Hex: #FF0000
GREEN - RGB: 65280 - Hex: #00FF00
BLUE - RGB: 255 - Hex: #0000FF
Process finished with exit code 0
```

En la enumeración simple el código define una enumeración simple en Kotlin con tres constantes: RED, GREEN y BLUE, donde cada constante es un objeto y se separan por comas.

The screenshot shows the Android Studio interface with the project 'AK\_Cap10\_11\_12' open. The code editor displays a Kotlin file named 'Cap12\_Enumaracion\_Simple.kt'. The code defines a simple enum class 'ColorEnum' with three entries: RED, GREEN, and BLUE. A main function demonstrates the use of the enum by printing its values and comparing them. The terminal output shows the execution of the code, including the enum values and the result of the comparison.

```
import java.nio.charset.StandardCharsets

// Definimos una enumeración simple de colores
enum class ColorEnum {
    RED, GREEN, BLUE
}

// Función principal para demostrar el uso de enums
fun main() {
    println("Lista de colores:")
    for (color in ColorEnum) {
        println("$color - Posición: ${color.ordinal}")
    }

    println("\nColor obtenido por valueOf('GREEN'):")
    val green = ColorEnum.valueOf("GREEN")
    println("Nombre: $green, Posición: ${green.ordinal}")

    println("\nComparación de colores:")
    println("¿RED es menor que BLUE? ${RED.ordinal < BLUE.ordinal}")

    Process finished with exit code 0
}
```

```
*C:\Program Files\Android\Android Studio\jbr\bin\java.exe* ...
Lista de colores:
RED - Posición: 0
GREEN - Posición: 1
BLUE - Posición: 2

Color obtenido por valueOf('GREEN'):
Nombre: GREEN, Posición: 1

Comparación de colores:
¿RED es menor que BLUE? true
Process finished with exit code 0
```

En cuanto al tema de mutabilidad el código muestra cómo las enumeraciones mutables en Kotlin pueden tener propiedades modificables. En el ejemplo, la propiedad population de la constante MARS en la enumeración Planet se puede cambiar después de su inicialización.

The screenshot shows the Android Studio interface with the project 'AK\_Cap10\_11\_12' open. The code editor displays a Kotlin file named 'Cap12\_Mutabilidad.kt'. The code defines an enum class 'Planet' with a population value of 0. It then prints the initial population of Mars and changes it to 3, printing the updated value.

```
enum class Planet(var population: Int) { MARS }

fun main() {
    // Imprimimos el valor initial de MARS
    println(Planet.MARS) // MARS[population=0]

    // Modificamos la población de Marte
    Planet.MARS.population = 3

    // Imprimimos el valor actualizado de MARS
    println(Planet.MARS) // MARS[population=3]
}
```

Los capítulos del 13 al 16 se muestran a continuación:

The screenshot shows the Android Studio interface with the project 'AK\_Cap13\_14\_15' open. The code editor displays a Kotlin file named 'Cap13\_7a.kt'. The code defines a class 'Persona3' with properties 'name' and 'age'. It then creates a list of four 'Persona3' objects and prints their names.

```
package com.example.ak_cap13_14_16

// 7a - Asigne nombres, ñáñase junto con delimitador
data class Persona3(val name: String, val age: Int)

fun main() {
    val persons = listOf(
        Persona3(name: "Max", age: 33),
        Persona3(name: "Peter", age: 23),
        Persona3(name: "Pamela", age: 18),
        Persona3(name: "David", age: 10)
    )
}
```

The screenshot shows the Android Studio interface with the project tree on the left and the code editor on the right. The code editor displays the following Kotlin code:

```
data class Persona4(val name: String, val age: Int)

fun main() {
    val persons = listOf(
        Persona4("Tod", 5),
        Persona4("Max", 33),
        Persona4("Frank", 13),
        Persona4("Peter", 80),
        Persona4("Pamela", 18)
    )
}
```

The output window shows the result of running the code:

```
*C:\Program Files\Android\Android Studio\jbr\bin\java.exe* ...
SummaryStatisticsInt(count=5, sum=149, min=5, max=80, avg=29.8)

Process finished with exit code 0
```

En el primero se define una clase Persona3 con nombre y edad. Luego, en la función main, crea una lista de personas, toma solo los nombres, los convierte a mayúsculas y los une en una sola cadena separados por | para finalmente imprimir esa cadena.

En el segundo se calculan estadísticas (conteo, suma, mínimo, máximo y promedio) sobre las edades de una lista de personas.

The screenshot shows the Android Studio interface with the project tree on the left and the code editor on the right. The code editor displays the following Kotlin code:

```
package com.example.ak_cap13_14_16

//Ejemplo #6: Agrupar personas por edad, mostrar la edad y los nombres juntos
data class Persona2(val name: String, val age: Int)

fun main() {
    val persons = listOf(
        Persona2("Tod", 5),
        Persona2("Max", 33),
        Persona2("Frank", 13),
        Persona2("Peter", 80),
        Persona2("Pamela", 18)
    )
}
```

The output window shows the result of running the code:

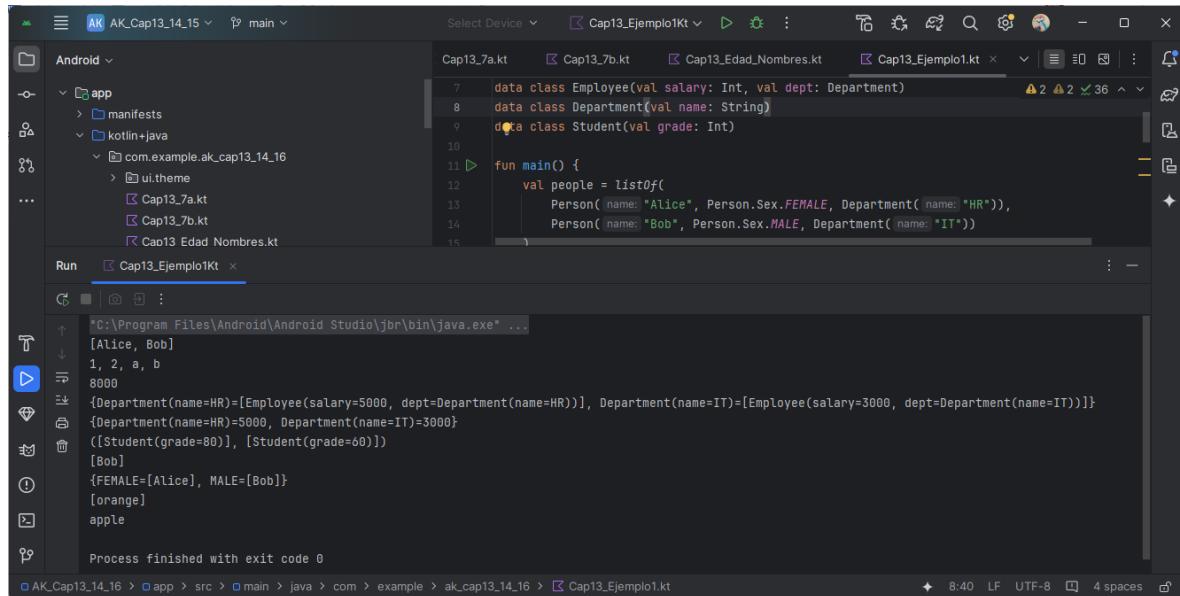
```
*C:\Program Files\Android\Android Studio\jbr\bin\java.exe* ...
{5=Tod, 33=Max, 13=Frank, 80=Peter, 18=Pamela}

Process finished with exit code 0
```

Por otra parte, este código que se muestra en la imagen de arriba agrupa una lista de personas según su edad y muestra, para cada edad, los nombres de las personas que tienen esa edad. Primero se define la clase Persona2 con las propiedades name y age. Luego se crea una lista de objetos Persona2. Con groupBy { it.age }, el código agrupa a las personas por su edad, generando un mapa donde la clave es la edad y el valor es una lista de personas con esa edad. Después,

con mapValues, se transforma cada lista de personas en una cadena de nombres unidos por punto y coma (;). Finalmente, se imprime el mapa resultante, mostrando por cada edad, los nombres de las personas correspondientes en formato de texto.

En el siguiente código se muestra una serie de ejemplos prácticos de operaciones comunes con colecciones en Kotlin, usando clases de ejemplo como Person, Employee, Department y Student. Primero, define varias clases de datos y una enumeración para el género. Luego, dentro de la función main, se ejecutan 10 operaciones diferentes.



```
data class Employee(val salary: Int, val dept: Department)
data class Department(val name: String)
data class Student(val grade: Int)

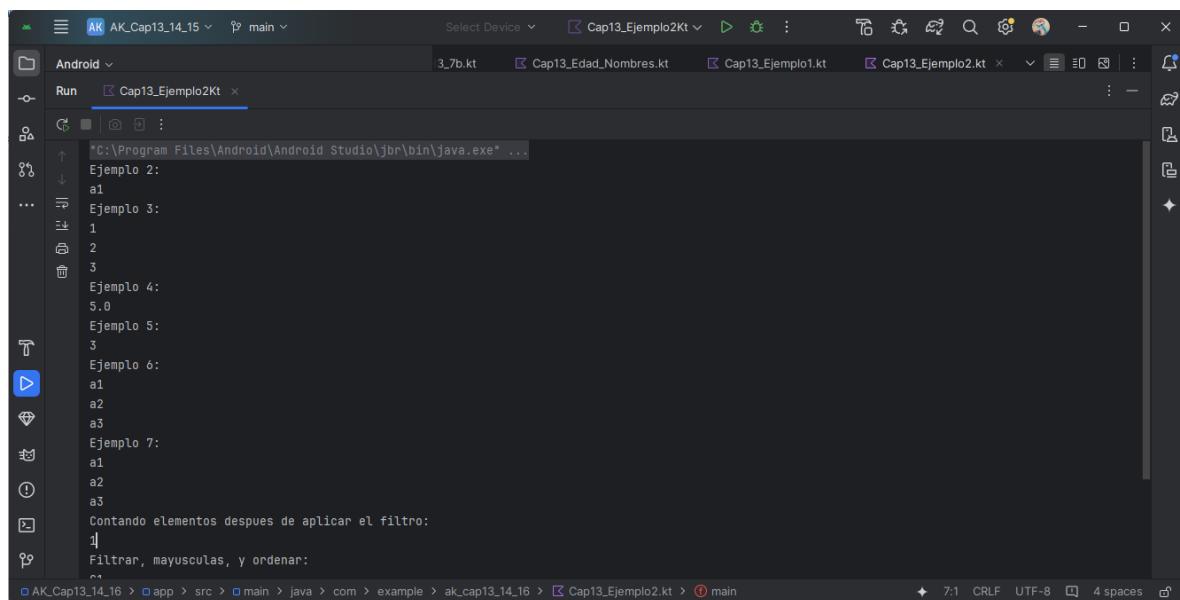
fun main() {
    val people = listOf(
        Person(name: "Alice", Person.Sex.FEMALE, Department(name: "HR")),
        Person(name: "Bob", Person.Sex.MALE, Department(name: "IT"))
    )
}
```

The terminal output shows the execution of Java code that prints a list of people and their details:

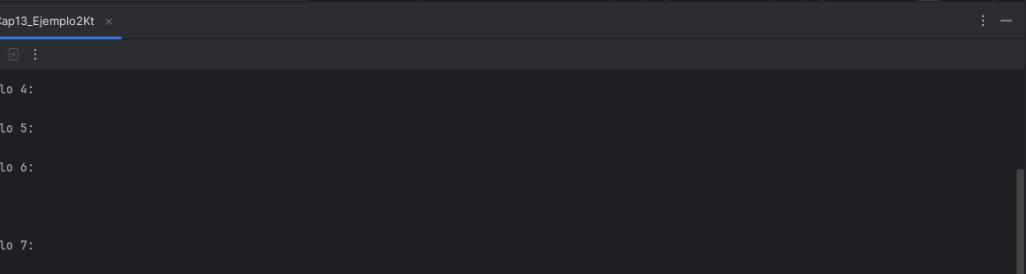
```
*C:\Program Files\Android\Android Studio\jbr\bin\java.exe* ...
[Alice, Bob]
1, 2, a, b
8000
{Department(name=HR)=[Employee(salary=5000, dept=Department(name=HR)), Department(name=IT)=[Employee(salary=3000, dept=Department(name=IT))]}
{Department(name=HR)=5000, Department(name=IT)=3000}
([Student(grade=80)], [Student(grade=60)])
[Bob]
{FEMALE=[Alice], MALE=[Bob]}
[orange]
apple

Process finished with exit code 0
```

En cambio, en el siguiente código se demuestran varios ejemplos de uso de colecciones y secuencias (streams) en Kotlin para transformar, filtrar y procesar datos de forma tanto perezosa (lazy) como ansiosa (eager).



```
Ejemplo 2:
a1
...
Ejemplo 3:
1
2
3
Ejemplo 4:
5.0
Ejemplo 5:
3
Ejemplo 6:
a1
a2
a3
Ejemplo 7:
a1
a2
a3
Contando elementos despues de aplicar el filtro:
4
Filtrar, mayusculas, y ordenar:
aa
```



```
Ejemplo 4:  
5.0  
Ejemplo 5:  
3  
Ejemplo 6:  
a1  
a2  
a3  
Ejemplo 7:  
a1  
a2  
a3  
Contando elementos despues de aplicar el filtro:  
1  
Filtrar, mayusculas, y ordenar:  
C1  
C2  
Ejemplo 1:  
a1  
  
Process finished with exit code 0
```

A continuación, también se muestra cómo se crea una lista de personas con nombre y edad, y luego genera una frase en formato de texto que indica quiénes son mayores de edad (18 años o más).

Primero, se filtra la lista para quedarse solo con las personas que tienen 18 años o más. Luego, se extraen los nombres de esas personas y los une en una sola cadena, usando " and " como separador, "In Germany" como prefijo y " are of legal age." como sufijo. El resultado es una oración amigable que se imprime con los nombres concatenados de los adultos.

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

- Project Structure:** The left sidebar shows the project structure under "Android". The "app" module contains "manifests", "kotlin+java", and several Kotlin files: "Cap13\_7a.kt", "Cap13\_7b.kt", "Cap13\_Edad\_Nombres.kt", "Cap13\_Ejemplo1.kt", "Cap13\_Ejemplo2.kt", "Cap13\_Mayor\_CadenaSalida.kt" (which is selected), "Cap14\_Expresion.kt", "Cap14\_MultExc.kt", and "Cap14\_Noexc.kt".
- Code Editor:** The main editor window displays the code for "Cap13\_Mayor\_CadenaSalida.kt".

```
1 package com.example.ak_cap13_14_16
2
3 // Ejemplo #5: Encontrar personas mayores de edad, una cadena con formato de salida
4 data class Persona(val name: String, val age: Int)
5
6 fun main() {
7     val persons = listOf(
8         Persona(name = "Tod", age = 5),
9         Persona(name = "Max", age = 33),
10        Persona(name = "Frank", age = 13),
11        Persona(name = "Peter", age = 80),
12        Persona(name = "Pamela", age = 18)
13    )
14
15     val phrase = persons
16 }
```
- Run Tab:** The bottom tab bar shows the selected run configuration: "Cap13\_Mayor\_CadenaSalidaKt".
- Output Window:** The bottom right window shows the terminal output:

```
C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
In Germany Max and Peter and Pamela are of legal age.

Process finished with exit code 0
```

En cuanto al capítulo 14 se muestran algunas excepciones:

The screenshot shows the Android Studio interface with the code editor open. The project navigation bar at the top shows 'AK Cap13\_14\_15' and 'main'. The code editor has three tabs: 'Cap13\_Mayor\_CadenaSalida.kt', 'Cap14\_MultExc.kt', and 'Cap14\_Expresion.kt'. The 'Cap14\_Expresion.kt' tab is active, displaying the following Kotlin code:

```
fun main() {
    // ...
    getString() // Si no hay excepciones, se devuelve el valor
} catch (e: Exception) {
    null // Si ocurre una excepción, retornamos null
}

// Imprime el resultado de la expresión try
println(s) // Imprime el valor o null dependiendo si hubo una excepción
}
```

The code editor's status bar at the bottom indicates '20:1 CRLF UTF-8 4 spaces'.

En esta primera imagen se demuestra cómo se puede usar try como una expresión (es decir, que devuelve un valor).

En la segunda imagen se muestra cómo capturar y manejar múltiples excepciones de manera eficiente usando bloques try, catch y finally.

The screenshot shows the Android Studio interface with the code editor open. The project navigation bar at the top shows 'AK Cap13\_14\_15' and 'main'. The code editor has three tabs: 'Cap13\_Mayor\_CadenaSalida.kt', 'Cap14\_MultExc.kt', and 'Cap14\_Expresion.kt'. The 'Cap14\_MultExc.kt' tab is active, displaying the following Kotlin code:

```
// Renombré la función 'cleanup' a 'cleanupResources' para evitar conflicto
fun cleanupResources() {
    // Código para limpiar recursos
    println("Limiando recursos...")
}

fun main() {
    try {
        doSomethingWithMultipleExceptions() // Intentamos hacer algo que puede generar una excepción
    } catch (e: IOException) {
        handleException(e) // Si ocurre una IOException, la manejamos
    } catch (e: Exception) {
        // Esto capture cualquier otra excepción
        handleException(e)
    } finally {
        cleanupResources() // Se ejecutará siempre, incluso si no hubo excepción
    }
}
```

The code editor's status bar at the bottom indicates '15:22 CRLF UTF-8 4 spaces'.

En la tercera podemos ver cómo leer el contenido de un archivo y manejar excepciones que podrían ocurrir durante la operación.

The screenshot shows the Android Studio interface with the project navigation bar at the top. Below it is the code editor with the file `Cap14_Noexc.kt` selected. The code implements a function `fileToString` that reads the content of a file into a string. A run configuration for `Cap14_NoexcKt` is selected in the toolbar. The output window shows an error message: "Ocurrio un error al leer el archivo: path/to/file.txt (El sistema no puede encontrar la ruta especificada)". The status bar at the bottom indicates the file is 26 lines long.

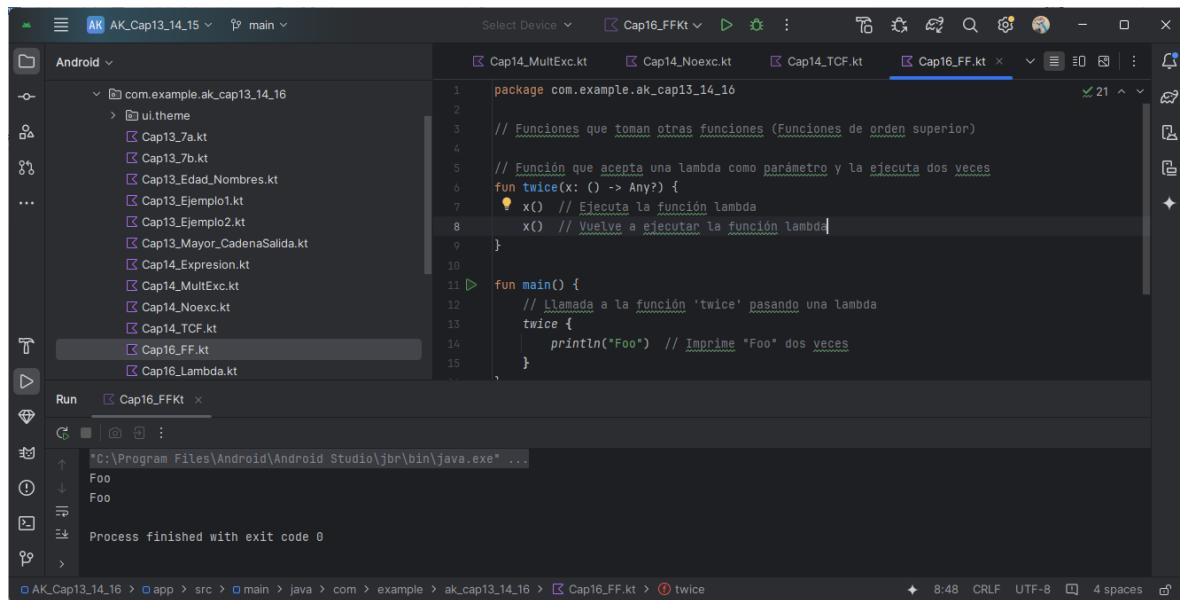
```
1 package com.example.ak_cap13_14_16
2
3 //Función que lanza una excepción (sin tener que manejarla explícitamente)
4 import ...
5
6 fun fileToString(file: File): String {
7     // Usamos FileInputStream para leer el contenido del archivo
8     val fileContent = file.inputStream().readBytes() // Lee todo el contenido como un arreglo de bytes
9     return String(fileContent) // Convierte el contenido de bytes a String
10 }
11
12
13 fun main() {
14     // Creamos un archivo de ejemplo (debes reemplazar esto por un archivo real si pruebas)
15     val file = File("path/to/file.txt")
16 }
```

En esta otra se muestra cómo manejar excepciones personalizadas utilizando un bloque try-catch-finally:

The screenshot shows the Android Studio interface with the project navigation bar at the top. Below it is the code editor with the file `Cap14_TCF.kt` selected. The code defines a custom exception `MyException`, handles it in the `main` function, and includes cleanup code. A run configuration for `Cap14_TCFKt` is selected in the toolbar. The output window shows the exception being caught and handled. The status bar at the bottom indicates the file is 26 lines long.

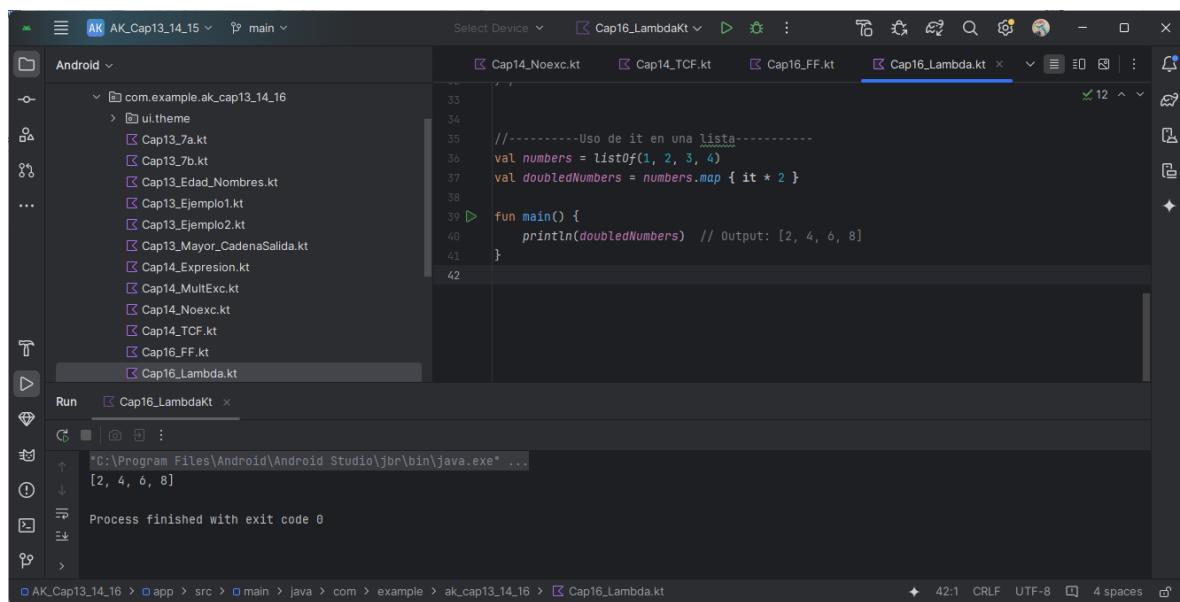
```
11 fun handle(e: MyException) {
12     // Manejo de la excepción
13     println("Excepcion capturada: ${e.message}")
14 }
15
16 fun cleanup() {
17     // Código para limpiar recursos
18     println("Limpiando recursos...")
19 }
20
21 fun main() {
22     try {
23         doSomething() // Intentamos hacer algo que puede generar una excepción
24     } catch (e: MyException) {
25         handle(e) // Si ocurre una MyException, la manejamos
26     }
27 }
```

En cuanto al capítulo 16 en la primera imagen se muestra cómo usar funciones de orden superior, que son funciones que aceptan otras funciones (en este caso, lambdas) como parámetros.



```
1 package com.example.ak_cap13_14_16
2
3 // Funciones que toman otras funciones (Funciones de orden superior)
4
5 // Función que acepta una lambda como parámetro y la ejecuta dos veces
6 fun twice(x: () -> Any?) {
7     x() // Ejecuta la función lambda
8     x() // Vuelve a ejecutar la función lambda
9 }
10
11 fun main() {
12     // Llamada a la función 'twice' pasando una lambda
13     twice {
14         println("Foo") // Imprime "Foo" dos veces
15     }
16 }
```

En el siguiente código podemos observar varios ejemplos de funciones lambda y su uso en diferentes contextos:



```
35
36 //-----Uso de it en una lista-----
37 val numbers = listOf(1, 2, 3, 4)
38 val doubledNumbers = numbers.map { it * 2 }
39
40 fun main() {
41     println(doubledNumbers) // Output: [2, 4, 6, 8]
42 }
```

```
1 package com.example.ak_cap13_14_16
2
3 //Funciones Lambda
4
5 //-----lambda con un solo argumento-----
6 val greet = { name: String ->
7     "Hello, $name!" // Devuelve un saludo con el nombre
8 }
9
10 fun main() {
11     println(greet("John")) // Output: Hello, John!
12 }
13
14
15 //-----lambda con múltiples argumentos-----
```

The screenshot shows the Android Studio interface with the Cap16\_Lambda.kt file open. The code defines a lambda function 'greet' that takes a single string argument and returns a greeting. It also contains a main function that prints the result of calling 'greet' with the argument 'John'. The output window shows the expected output: 'Hello, John!'. The status bar at the bottom indicates the file is saved and shows the current time as 12:2.

Se comentararon los demás para ejecutarlos según se requiera.

En cuanto al siguiente se muestra el uso de funciones en línea (inline) en Kotlin.

```
1 package com.example.ak_cap13_14_16
2
3 //Funciones en linea (inline)
4 //Declaración de una función en linea
5 inline fun sayHello(name: String) {
6     println("Hello, $name!")
7 }
8
9 fun main() {
10     sayHello(name = "Alice") // Imprime "Hello, Alice!" sin necesidad de una llamada de función
11 }
```

The screenshot shows the Android Studio interface with the Cap16\_Linea.kt file open. The code defines an inline function 'sayHello' that takes a string argument and prints a greeting. In the main function, it calls 'sayHello' with the argument 'Alice', demonstrating how inline functions can be used without a separate function call. The output window shows the expected output: 'Hello, Alice!'. The status bar at the bottom indicates the file is saved and shows the current time as 11:2.

En el siguiente código se demuestra el uso de referencias de funciones en Kotlin.

The screenshot shows the Android Studio interface with the code editor open. The file is `Cap16_Ref_Func.kt`. The code defines a package and a function `addTwo` that adds 2 to its parameter. It then uses this function in a `map` operation on a list of integers, printing the result. The run output shows the list [3, 4, 5, 6].

```
1 package com.example.ak_cap13_14_16
2
3 // Referencias de funciones
4
5 // Definición de la función que se va a referenciar
6 fun addTwo(x: Int) = x + 2
7
8 fun main() {
9     val numbers = listOf(1, 2, 3, 4)
10    // Usando la referencia de la función addTwo
11    val result = numbers.map(::addTwo)
12    println(result) // Output: [3, 4, 5, 6]
13 }
14
```

Y, por último, se muestra cómo sobrecargar operadores en Kotlin. En este caso, se sobrecarga el operador `[]` para una clase personalizada llamada `IntListWrapper`.

The screenshot shows the Android Studio interface with the code editor open. The file is `Cap16_Sob_Oper.kt`. The code defines a class `IntListWrapper` that overrides the `[]` operator to return the element at the specified index. In the `main` function, it creates an instance of `IntListWrapper` and prints the third element of a list [1, 2, 3, 4]. The run output shows the value 3.

```
1 package com.example.ak_cap13_14_16
2
3 // Sobre carga de operadores
4
5 // Sobre cargando el operador '[' para la clase IntListWrapper
6 data class IntListWrapper(val wrapped: List<Int>) {
7     operator fun get(position: Int): Int = wrapped[position]
8 }
9
10 fun main() {
11     val listWrapper = IntListWrapper(listOf(1, 2, 3, 4))
12     // Usando el operador sobre cargado para acceder a los elementos de la lista
13     println(listWrapper[2]) // Output: 3 (Accede al tercer elemento de la lista)
14 }
```

En cuanto al [\*\*capítulo 17\*\*](#) podemos ver qué en el primer código se muestran dos formas equivalentes de declarar funciones en Kotlin que no devuelven ningún valor (es decir, funciones cuyo tipo de retorno es `Unit`, el equivalente a `void` en otros lenguajes como Java):

```
1 package com.example.ak_cap17_18_19_20.ui.theme
2 // Ejemplo con Unit declarado
3 fun printHelloWithUnit(name: String?): Unit {
4     if (name != null)
5         println("Hello $name")
6 }
7
8 // Ejemplo sin Unit declarado (equivalente)
9 fun printHello(name: String?) {
10     if (name != null)
11         println("Hello $name")
12 }
13
14 fun main() {
15     printHelloWithUnit("Hello *Carlos")
```

C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
Hello Carlos
Hello Maria
Process finished with exit code 0

En el otro código se muestra cómo se pueden declarar funciones en Kotlin con tipo de retorno explícito o con tipo inferido automáticamente por el compilador:

```
1 package com.example.ak_cap17_18_19_20
2
3 // Función con tipo de retorno explícito
4 fun doubleExplicit(x: Int): Int = x * 2
5
6 // Función con tipo inferido
7 fun doubleInferred(x: Int) = x * 2
8
9 fun main() {
10     println("Doble explicito: ${doubleExplicit(4)}") // Output: 8
11     println("Doble inferido: ${doubleInferred(5)}") // Output: 10
12 }
```

C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
Doble explicito: 8
Doble inferido: 10
Process finished with exit code 0

En este capítulo también se demuestra la diferencia entre == y === en Kotlin al comparar variables tipo String:

The screenshot shows the Android Studio interface with the project structure on the left and a code editor on the right. The code editor displays a file named `Cap17_Comparacion.kt` with the following content:

```
1 package com.example.ak_cap17_18_19_20
2
3 fun main() {
4     val x = "Hola"
5     val y = "Hola"
6     val z = x
7
8     println("x == y: ${x == y}")      // true, compara contenido
9     println("x === y: ${x === y}")    // Puede ser true o false (depende de interning)
10
11    println("x === z: ${x === z}")   // true, misma referencia
12 }
```

The run tab at the bottom shows the output of the program:

```
x == y: true
x === y: true
x === z: true
Process finished with exit code 0
```

Además, podemos ver una demostración sencilla del uso de interpolación de cadenas en Kotlin.

The screenshot shows the Android Studio interface with the project structure on the left and a code editor on the right. The code editor displays a file named `Cap17_Cadena.kt` with the following content:

```
1 package com.example.ak_cap17_18_19_20
2
3 fun main() {
4     val num = 10
5     val text = "i = $num"
6     println(text) // Output: i = 10
7 }
```

The run tab at the bottom shows the output of the program:

```
i = 10
Process finished with exit code 0
```

También se demuestra cómo Kotlin maneja la nulabilidad usando tipos no anulables (String) y anulables (String?):

The screenshot shows the Android Studio interface with the project tree on the left and a code editor on the right. The code editor displays the following Kotlin code:

```
1 package com.example.ak_cap17_18_19_20
2
3 fun main() {
4     // Referencia no anulable
5     var a: String = "abc"
6     // a = null // Esto genera error de compilación
7
8     // Referencia anulable
9     var b: String? = "abc"
10    b = null // Esto si se permite
11
12    println("a: $a")
13    println("b: $b")
14 }
15
```

The code demonstrates nullable and non-nullable references. It prints "a: abc" and "b: null". The run log shows the output and indicates the process finished with exit code 0.

En cuanto al **capítulo 18** se muestran ejemplos de cómo trabajar con rangos en Kotlin, incluyendo rangos ascendentes, descendentes, con pasos y el uso de la función until.

The screenshot shows the Android Studio interface with the project tree on the left and a code editor on the right. The code editor displays the following Kotlin code:

```
*C:\Program Files\Android\Android Studio\jbr\bin\java.exe* ...
? Rango ascendente (1..4):
1234

? Rango descendente (4..1):
4321

? Rango descendente con downTo (4 downTo 1):
4321

? Rango con paso step (1..4 step 2):
13

? Rango descendente con paso (4 downTo 1 step 2):
42

? Rango hasta con until (1 until 10):
123456789

Process finished with exit code 0
```

The code demonstrates various range operations: ascending ranges (1..4), descending ranges (4..1), descending ranges with downTo (4 downTo 1), step ranges (1..4 step 2), descending step ranges (4 downTo 1 step 2), and ranges with until (1 until 10). The run log shows the output and indicates the process finished with exit code 0.

En el **capítulo 19** se ejemplifican conceptos de variación de tipo en Kotlin, específicamente variación del sitio de la declaración, variación del sitio de uso y proyección de estrellas.

The screenshot shows the Android Studio interface. The top navigation bar includes 'AK AK\_Cap17\_18\_19\_20', 'Version control', 'Medium Phone API 35', and tabs for 'Cap19Kt', 'Cap17\_Comparacion.kt', 'Cap18.kt', and 'Cap19.kt'. The left sidebar shows the project structure under 'Android' with 'app' selected, containing 'manifests', 'kotlin+java', and 'com.example.ak\_cap17\_18\_19\_20' with files 'ui.theme', 'Cap17\_Anulable.kt', and 'Cap17\_Cardenas.kt'. The main code editor shows the following Kotlin code:

```
43    }
44    // -----
45    fun starProjectionExample() {
46        println("\n> Proyección de estrellas")
47
48        val starList: MutableList<String> = mutableListOf("X", "X", "X", "X", "X")
49
50        starList.forEach { println(it) }
51    }
52
53    companion object {
54        const val TAG = "Cap19Kt"
55    }
56}
```

The 'Run' tab is active, showing the command 'C:\Program Files\Android\Android Studio\jbr\bin\java.exe ...'. The output window below shows the execution results:

```
*C:\Program Files\Android\Android Studio\jbr\bin\java.exe* ...
? Variacion del sitio de la declaracion
Consuming: Hola Mundo
?
? Variacion del sitio de uso
Leido desde out-list: Uno
Tamaño tras agregar a in-list: 4
Valor leido de in-list: A
?
? Proyección de estrellas
Elemento de lista con *: X
?
Process finished with exit code 0
```

En el [\*\*capítulo 20\*\*](#) se muestran ejemplos de diferentes conceptos de herencia, anulación y clases en Kotlin.

The screenshot shows the Android Studio interface. The top navigation bar includes 'AK AK\_Cap17\_18\_19\_20', 'Version control', 'Medium Phone API 35', and tabs for 'Cap20Kt', 'Cap17\_Comparacion.kt', 'Cap18.kt', and 'Cap19.kt'. The left sidebar shows the project structure under 'Android' with 'app' selected, containing 'manifests'. The main code editor shows the following Kotlin code:

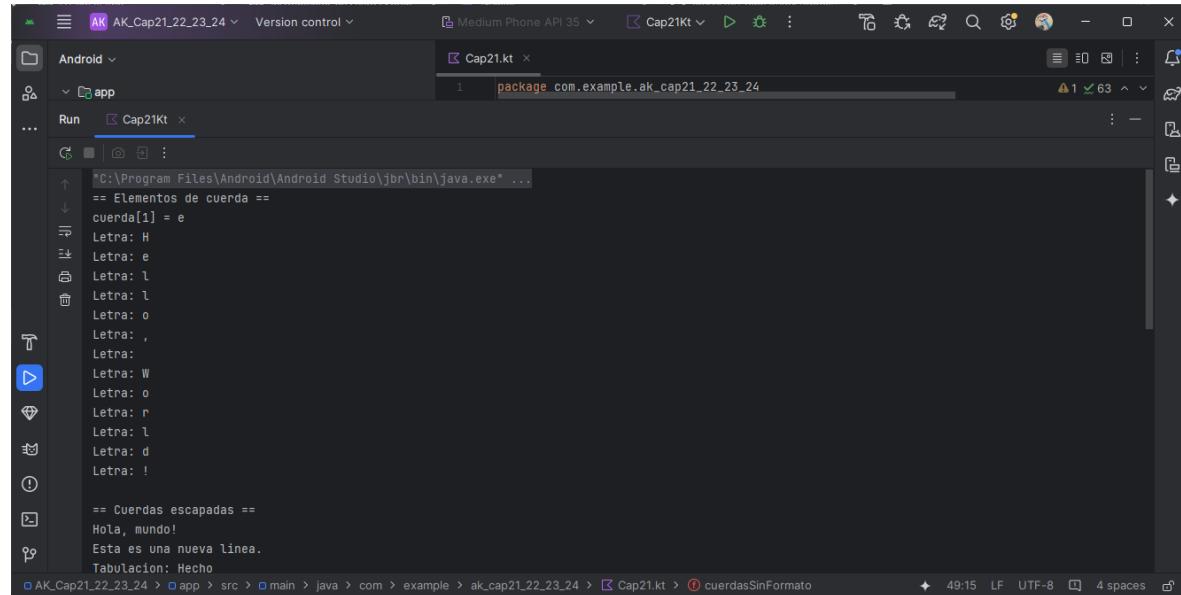
```
83     println("Error: ${e.message}")
84 }
```

The 'Run' tab is active, showing the command 'C:\Program Files\Android\Android Studio\jbr\bin\java.exe ...'. The output window below shows the execution results:

```
*C:\Program Files\Android\Android Studio\jbr\bin\java.exe* ...
? Herencia de campos
x is equal to 10
?
? Herencia de metodos
Jumping...
Sneaking around...
?
Override de propiedades
Error: The car is broken
?
Override de metodos en interfaz
Puede navegar Titanic?: true
Sailing... now sinking.
Sinking the Titanic!
Puede navegar Titanic despues?: false
?
Process finished with exit code 0
```

En lo que respecta a los [\*\*capítulos 21, 22, 23 y 24\*\*](#) en el capítulo 21 se demuestran diferentes formas de trabajar con cuerdas (strings) en el lenguaje. Se organiza en funciones que se ejecutan desde `main()` para mostrar ejemplos prácticos. Primero, la función `elementosDeCuerda()` enseña cómo acceder e iterar los caracteres de una cuerda usando índices y bucles. Luego, `cuerdasEscapadas()` muestra el uso de caracteres especiales dentro de cuerdas usando secuencias de escape como `\n`, `\t` y `\$`, además de incluir un símbolo Unicode. La función `cuerdasSinFormato()` presenta las cuerdas sin formato (también llamadas literales crudos), que permiten escribir varias líneas sin necesidad de escapes, y cómo limpiar la indentación con

`trimMargin()`. La sección `plantillasDeCuerda()` demuestra cómo insertar variables o expresiones dentro de una cuerda usando el símbolo \$, incluyendo la forma de representar un signo de dólar literal. Por último, `igualdadDeCuerdas()` compara cuerdas en cuanto a su contenido (==) y su referencia en memoria (==).



A screenshot of the Android Studio interface. The left sidebar shows the project structure for 'AK\_Cap21\_22\_23\_24' with an 'app' module selected. The main editor window displays a Kotlin file named 'Cap21.kt'. The code in the editor is:

```
package com.example.ak_cap21_22_23_24

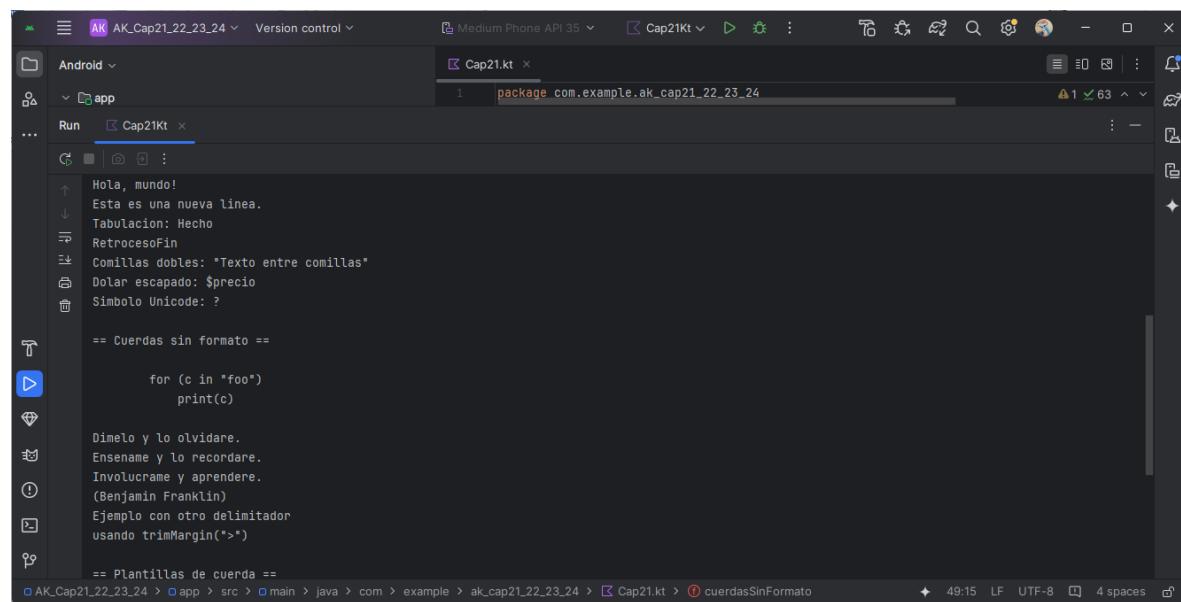
// Cuerdas con formato
// == Elementos de cuerda ==
// cuerda[1] = e
// Letra: H
// Letra: e
// Letra: l
// Letra: l
// Letra: o
// Letra: ,
// Letra: !
// Letra: W
// Letra: o
// Letra: r
// Letra: l
// Letra: d
// Letra: !

// == Cuerdas escapadas ==
Hola, mundo!
Esta es una nueva linea.
Tabulacion: Hecho

// Cuerdas sin formato ==
for (c in "foo")
    print(c)

Dimelo y lo olvidare.
Ensename y lo recordare.
Involucrame y aprendere.
(Benjamin Franklin)
Ejemplo con otro delimitador
usando trimMargin(">")

// Plantillas de cuerda ==
```



A screenshot of the Android Studio interface, similar to the previous one but with different code in the editor. The main editor window displays a Kotlin file named 'Cap21.kt'. The code in the editor is:

```
package com.example.ak_cap21_22_23_24

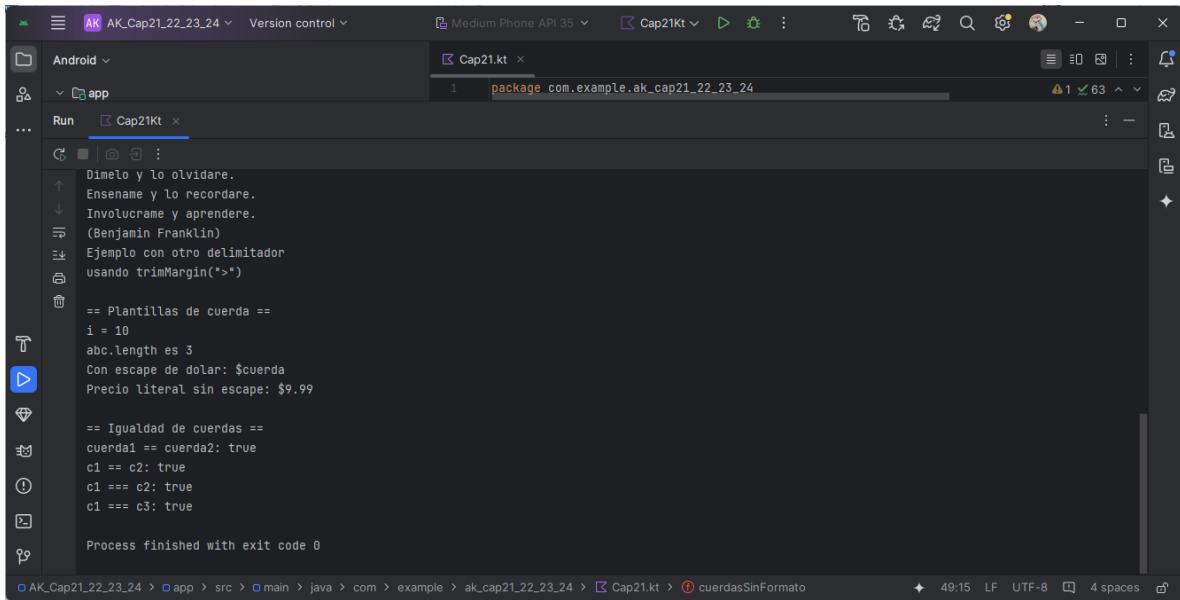
// Cuerdas con formato
// == Elementos de cuerda ==
// cuerda[1] = e
// Letra: H
// Letra: e
// Letra: l
// Letra: l
// Letra: o
// Letra: ,
// Letra: !
// Letra: W
// Letra: o
// Letra: r
// Letra: l
// Letra: d
// Letra: !

// == Cuerdas escapadas ==
Hola, mundo!
Esta es una nueva linea.
Tabulacion: Hecho

// Cuerdas sin formato ==
for (c in "foo")
    print(c)

Dimelo y lo olvidare.
Ensename y lo recordare.
Involucrame y aprendere.
(Benjamin Franklin)
Ejemplo con otro delimitador
usando trimMargin(">")

// Plantillas de cuerda ==
```



```
AK Cap21_22_23_24 Version control
Medium Phone API 35 Cap21Kt
Run Cap21Kt
...
Dimelo y lo olvidare.
Ensename y lo recordare.
Involucrame y aprendere.
(Benjamin Franklin)
Ejemplo con otro delimitador
usando trimMargin(">")

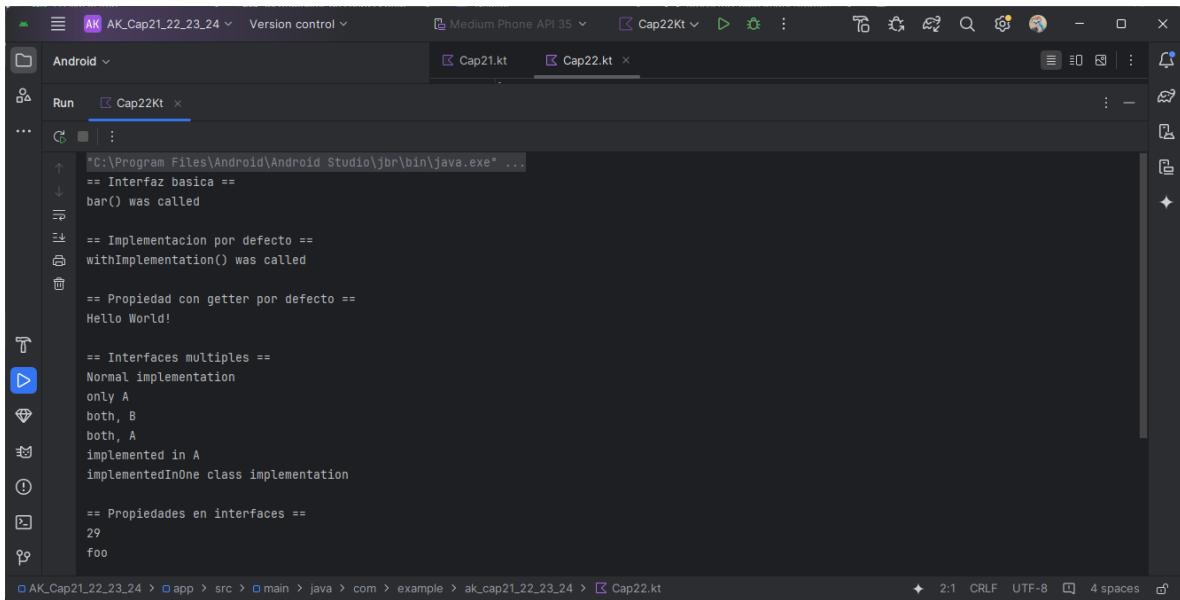
== Plantillas de cuerda ==
i = 10
abc.length es 3
Con escape de dolar: $cuerda
Precio literal sin escape: $9.99

== Igualdad de cuerdas ==
cuerdal == cuerda2: true
c1 == c2: true
c1 === c2: true
c1 === c3: true

Process finished with exit code 0
```

AK\_Cap21\_22\_23\_24 > app > src > main > java > com > example > ak\_cap21\_22\_23\_24 > Cap21.kt > cuerdaSinFormato

En cuanto al capítulo 22 se muestra cómo usar interfaces, incluyendo la declaración de métodos, implementaciones predeterminadas, propiedades con getter, y cómo manejar conflictos cuando se implementan múltiples interfaces con métodos similares. También se explica cómo usar la palabra clave “super” para acceder a implementaciones predeterminadas en interfaces y resolver conflictos de implementación entre interfaces. Todo esto se demuestra en la función main().

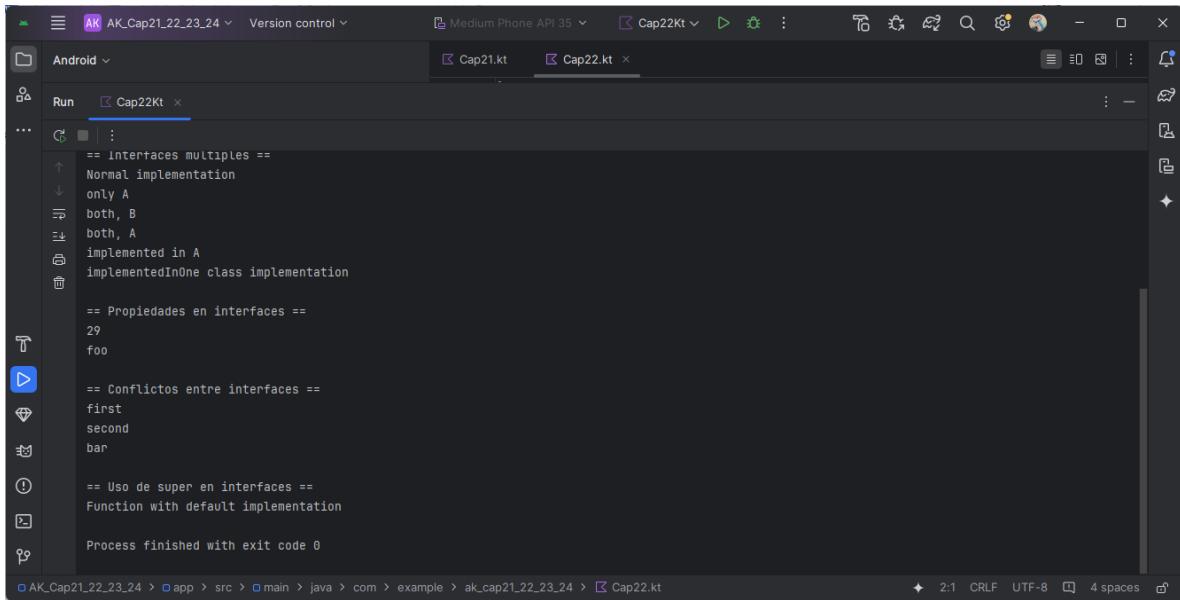


```
AK Cap21_22_23_24 Version control
Medium Phone API 35 Cap22Kt
Run Cap22Kt
...
*C:\Program Files\Android\Android Studio\jbr\bin\java.exe* ...
== Interfaz basica ==
bar() was called
== Implementacion por defecto ==
withImplementation() was called
== Propiedad con getter por defecto ==
Hello World!

== Interfaces multiples ==
Normal implementation
only A
both, B
both, A
implemented in A
implementedInOne class implementation

== Propiedades en interfaces ==
29
foo
```

AK\_Cap21\_22\_23\_24 > app > src > main > java > com > example > ak\_cap21\_22\_23\_24 > Cap22.kt



```
== Interfaces multiples ==
only A
both, B
both, A
implemented in A
implementedInOne class implementation

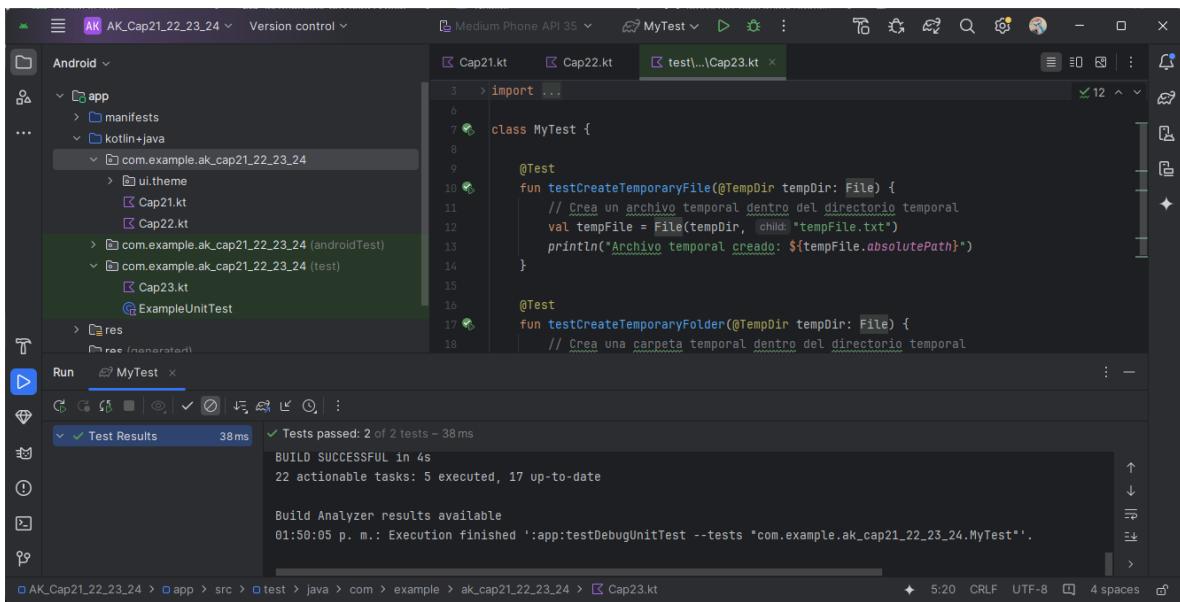
== Propiedades en interfaces ==
29
foo

== Conflictos entre interfaces ==
first
second
bar

== Uso de super en interfaces ==
Function with default implementation

Process finished with exit code 0
```

En el capítulo 23 definimos una clase de prueba llamada MyTest usando JUnit 5, la cual tiene dos métodos marcados con @Test que prueban la creación de archivos y carpetas temporales usando @TempDir, una anotación que proporciona un directorio temporal automático para las pruebas. En cuanto a testCreateTemporaryFile esta crea un archivo llamado tempFile.txt dentro del directorio temporal, mientras que testCreateTemporaryFolder crea una carpeta llamada tempFolder en ese mismo directorio.



```
import ...

class MyTest {

    @Test
    fun testCreateTemporaryFile(@TempDir tempDir: File) {
        // Crea un archivo temporal dentro del directorio temporal
        val tempFile = File(tempDir, "tempfile.txt")
        println("Archivo temporal creado: ${tempfile.getAbsolutePath}")
    }

    @Test
    fun testCreateTemporaryFolder(@TempDir tempDir: File) {
        // Crea una carpeta temporal dentro del directorio temporal
    }
}
```

En el capítulo 24 se muestra cómo Kotlin simplifica muchas tareas comunes frente a Java. Usa “val” y “var” para declarar variables, es seguro ante nulos, y permite comparar igualdad de contenido (==) e identidad (==>). Las expresiones “if” y “try” devuelven valores, no son solo estructuras de control. También se crean objetos sin

“new”, se aprovechan las data class para generar métodos útiles automáticamente, y se accede fácilmente a mapas con [].

The screenshot shows the Android Studio interface. The left sidebar shows the project structure under 'Android' with 'app' selected. The main area has four tabs: Cap21.kt, Cap22.kt, Cap23.kt, and Cap24.kt. Cap24.kt is the active tab, displaying the following Kotlin code:

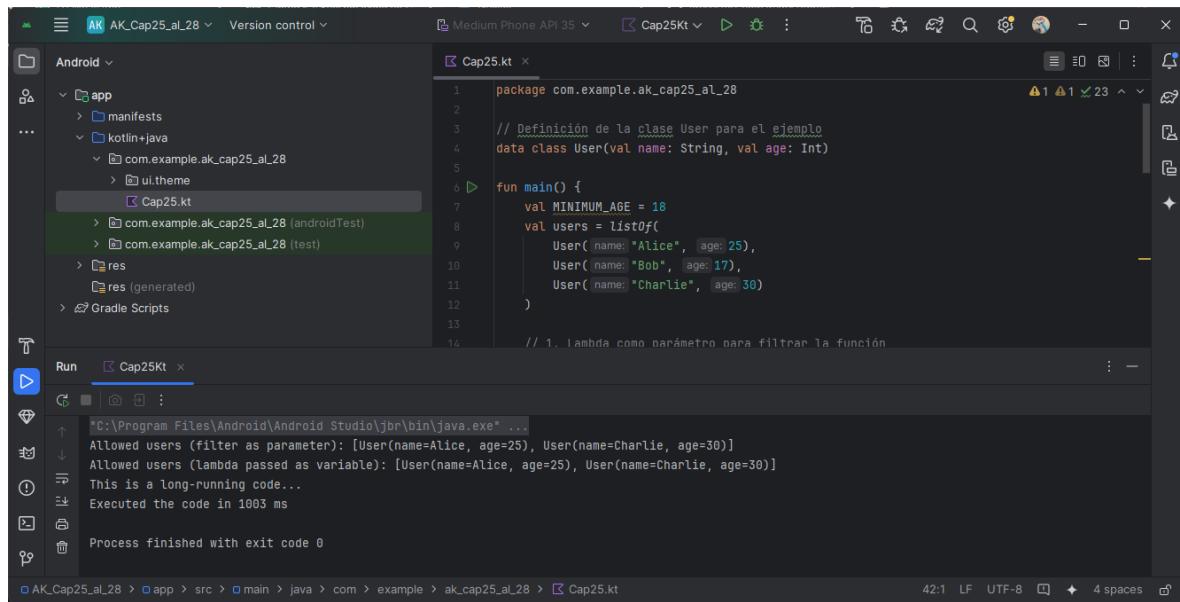
```
1 package com.example.ak_cap21_22_23_24
2
3 fun main() {
4     // Declaración de variables
5     val finalValue: Int = 42           // Inmutable (como final en Java)
6     var mutableValue = 100             // Mutable con inferencia de tipo
7
8     println("Final: $finalValue, Mutable: $mutableValue")
}
```

Below the code editor is the 'Run' tool window, which shows the output of the program execution:

```
*C:\Program Files\Android\Android Studio\jbr\bin\java.exe* ...
Final: 42, Mutable: 100
Longitud del nombre (seguro para nulos): desconocido
a == b: true
a == b: true
a == c: true
Mayor de edad
Resultado del try: 123
Nombre: Karla, Edad: 22
Son iguales?: true
Valor en mapa['clave']: valor
Process finished with exit code 0
```

En los capítulos 25, 26, 27 y 28 en el capítulo 25 se muestra cómo utilizar funciones de orden superior y lambdas en diferentes contextos. Primero, se define una clase User con propiedades name y age para representar a los usuarios. Luego, en la función main, crea una lista de usuarios con diferentes edades y establece una edad mínima (18 años). Se utilizan tres enfoques para filtrar los usuarios mayores de edad:

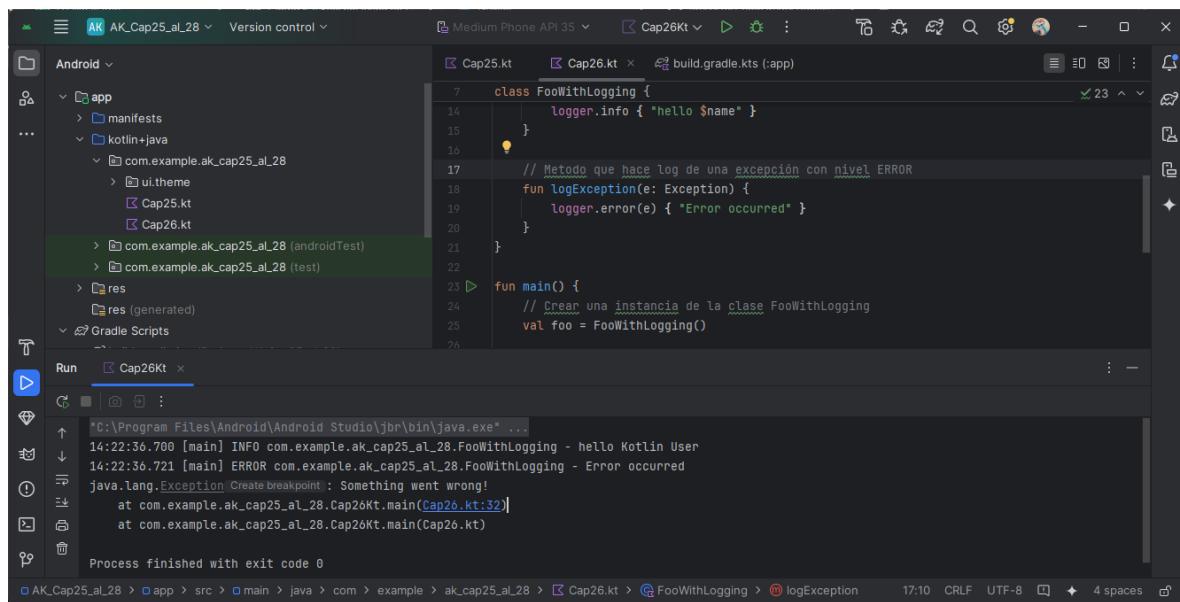
- Lambda como parámetro: Usa una lambda directamente dentro del método filter para filtrar la lista de usuarios.
- Lambda como variable: Define una lambda isOfAllowedAge que luego se pasa como argumento a filter para obtener los usuarios mayores de edad.
- Benchmarking con lambda: Utiliza un objeto anónimo con una función realtime para medir el tiempo de ejecución de un bloque de código (simulando un código de larga duración con Thread.sleep(1000)).



```
1 package com.example.ak_cap25_al_28
2
3 // Definición de la clase User para el ejemplo
4 data class User(val name: String, val age: Int)
5
6 fun main() {
7     val MINIMUM_AGE = 18
8     val users = listOf(
9         User(name = "Alice", age = 25),
10        User(name = "Bob", age = 17),
11        User(name = "Charlie", age = 30)
12    )
13
14    // 1. Lambda como parámetro para filtrar la función
15 }
```

The screenshot shows the Android Studio interface with the Cap25.kt file open in the editor. The code defines a User class and a main function that filters users based on age using a lambda expression. The Run tab shows the output of the code execution.

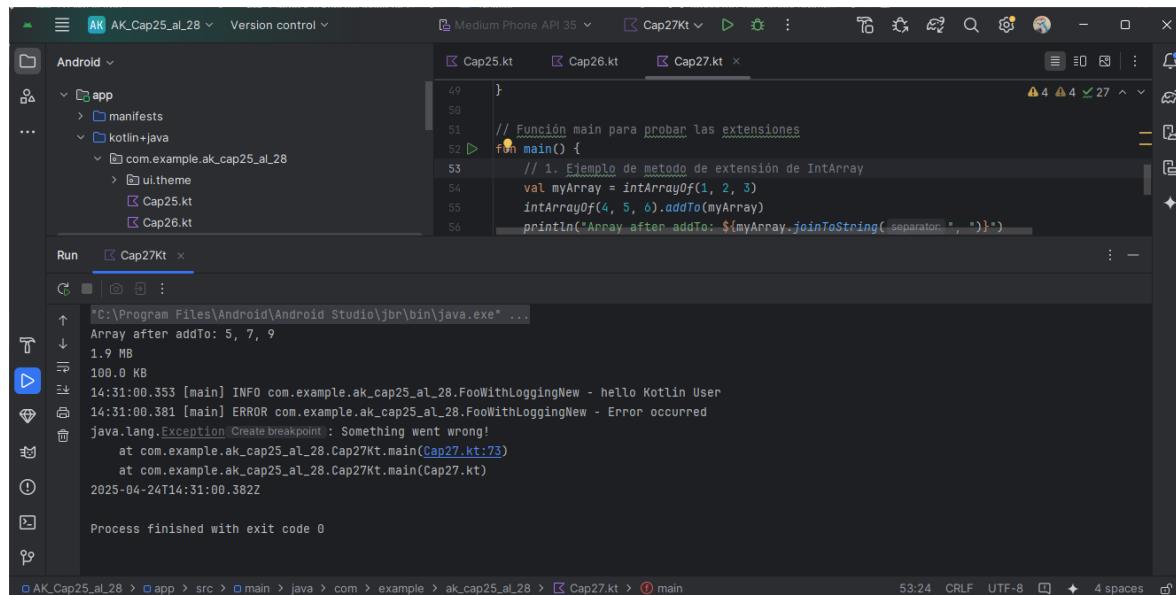
Por otra parte, en el capítulo 26 se utiliza la librería KLogging para implementar un sistema de logging en una clase. La clase FooWithLogging contiene un companion object que establece un logger estático, que se usa en los métodos de la clase. En el método bar, se registra un mensaje de nivel INFO que incluye el nombre del usuario. En el método logException, se captura una excepción y se registra con un mensaje de nivel ERROR. En la función main, se crea una instancia de FooWithLogging, se llama al método bar para registrar un mensaje y luego se simula una excepción que se pasa al método logException para ser registrada. Esto permite ver cómo se generan logs tanto para eventos informativos como para errores.



```
7 class FooWithLogging {
8     logger.info { "Hello $name" }
9 }
10
11 // Método que hace log de una excepción con nivel ERROR
12 fun logException(e: Exception) {
13     logger.error(e) { "Error occurred" }
14 }
15
16 fun main() {
17     // Crear una instancia de la clase FooWithLogging
18     val foo = FooWithLogging()
19
20     foo.bar("Kotlin User")
21
22     try {
23         throw java.lang.Exception("Something went wrong!")
24     } catch (e: Exception) {
25         foo.logException(e)
26     }
27 }
```

The screenshot shows the Android Studio interface with the Cap26.kt file open in the editor. The code defines a FooWithLogging class with methods for logging info and error messages, and a main function that creates an instance and calls these methods. The Run tab shows the output of the code execution, including log messages and exception details.

En el capítulo 27 se definen varias extensiones para IntArray, Long, File y Date. Permite agregar elementos de un arreglo a otro, convertir números a un formato legible por humanos, verificar y eliminar archivos, y formatear fechas en ISO. Además, utiliza KLogging para registrar mensajes y excepciones en la clase FooWithLoggingNew. El método main prueba estas extensiones con ejemplos prácticos.



```
49 }
50
51 // Función main para probar las extensiones
52 fun main() {
53     // 1. Ejemplo de método de extensión de IntArray
54     val myArray = intArrayOf(1, 2, 3)
55     intArrayOf(4, 5, 6).addTo(myArray)
56     println("Array after addTo: ${myArray.joinToString(", ")}")
```

The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right. The code editor displays a Kotlin file named Cap27Kt with the above code. The run tab at the bottom shows the terminal output:

```
[C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...]
Array after addTo: 5, 7, 9
1.9 MB
100.0 KB
14:31:00.353 [main] INFO com.example.ak_cap25_al_28.FooWithLoggingNew - hello Kotlin User
14:31:00.381 [main] ERROR com.example.ak_cap25_al_28.FooWithLoggingNew - Error occurred
java.lang.Exception: Create breakpoint : Something went wrong!
    at com.example.ak_cap25_al_28.Cap27Kt.main(Cap27.kt:73)
    at com.example.ak_cap25_al_28.Cap27Kt.main(Cap27.kt)
2025-04-24T14:31:00.382Z

Process finished with exit code 0
```

En el capítulo 28 vemos cómo usar los modificadores de visibilidad en Kotlin: “public” (accesible desde cualquier lugar), “private” (solo dentro de la clase), “protected” (dentro de la clase y sus subclases) e “internal” (dentro del mismo módulo). La clase VisibilityExample contiene propiedades con estos modificadores y la función showProperties las imprime. La subclase SubVisibilityExample accede a la propiedad “protected”.

The screenshot shows the Android Studio interface with the project 'AK\_AK\_Cap25\_aL\_28' open. The code editor displays 'Cap28.kt' with the following content:

```
class SubVisibilityExample : VisibilityExample() {
}
fun main() {
    // Creando un objeto de VisibilityExample
    val example = VisibilityExample()
    // Accediendo a las propiedades públicas e internas
    println("Public Name from main: ${example.publicName}")
    println("Internal Name from main: ${example.internalName}")
}
```

The 'Run' tab is selected, showing the output of the run command:

```
C:\Program Files\Android\Android Studio\jbr\bin\java.exe ...
Public Name from main: Avijit
Internal Name from main: Avijit
Public Name: Avijit
Private Name: Avijit
Protected Name: Avijit
Internal Name: Avijit
Protected Name from Subclass: Avijit
Process finished with exit code 0
```

Los capítulos 29, 30, 31, 32 y 33 se muestran a continuación, donde en el capítulo 29 primeramente se nos solicita el uso de la función “apply” en Kotlin para inicializar un objeto y realizar operaciones sobre él de forma concisa. En el bloque main(), se define una ruta de directorio como cadena (someDirectoryPath) y luego se crea un objeto File con esa ruta. Usando apply, se llama al método “mkdirs()” directamente sobre ese objeto File para crear el directorio, incluyendo cualquier directorio padre que no exista. La función apply permite ejecutar ese método dentro de su bloque y luego devuelve el mismo objeto, lo que facilita el encadenamiento o inicialización compacta. Finalmente, se imprime un mensaje indicando que el directorio ha sido creado.

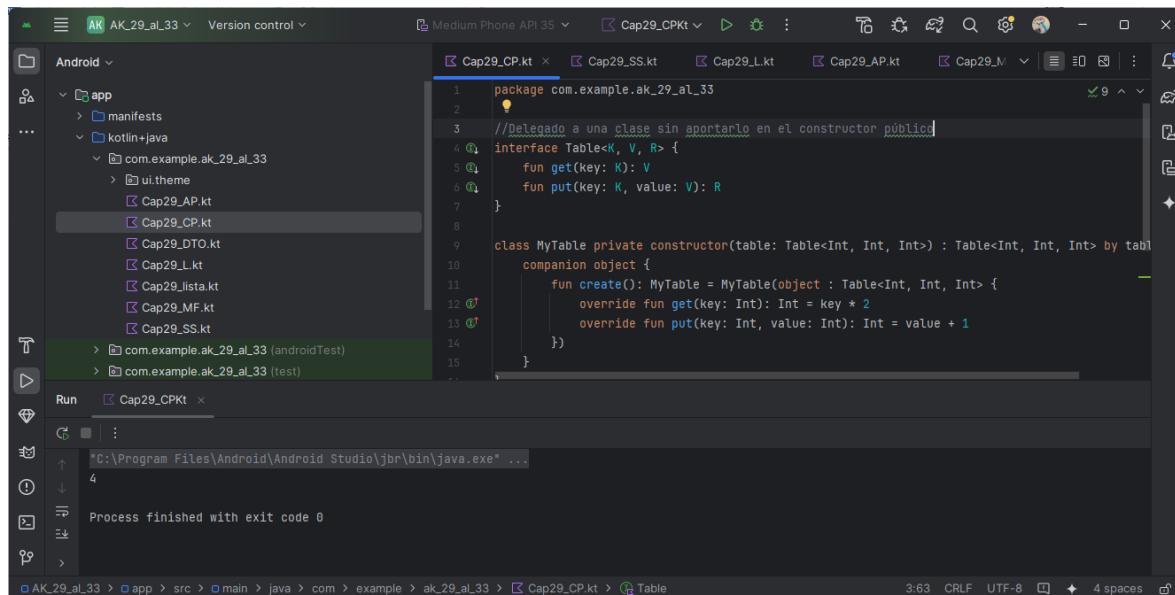
The screenshot shows the Android Studio interface with the project 'AK\_29\_aL\_33' open. The code editor displays 'Cap29\_AP.kt' with the following content:

```
package com.example.ak_29_aL_33
//Uso de apply para inicializar objetos o lograr encadenamiento de métodos
import java.io.File
fun main() {
    val dir = "someDirectoryPath"
    // Usando apply para crear un directorio
    File(dir).apply { mkdirs() }
    println("Directory created")
}
```

The 'Run' tab is selected, showing the output of the run command:

```
C:\Program Files\Android\Android Studio\jbr\bin\java.exe ...
Directory created
Process finished with exit code 0
```

Posteriormente se nos solicita usar delegación para implementar una interfaz sin exponer su implementación al público. La clase “MyTable” implementa la interfaz “Table” delegando su comportamiento a una instancia creada internamente con un constructor privado. El método create() en el companion object genera esa instancia. En main(), se crea una instancia con MyTable.create() y se prueba el método get(), que devuelve el doble de la clave dada.

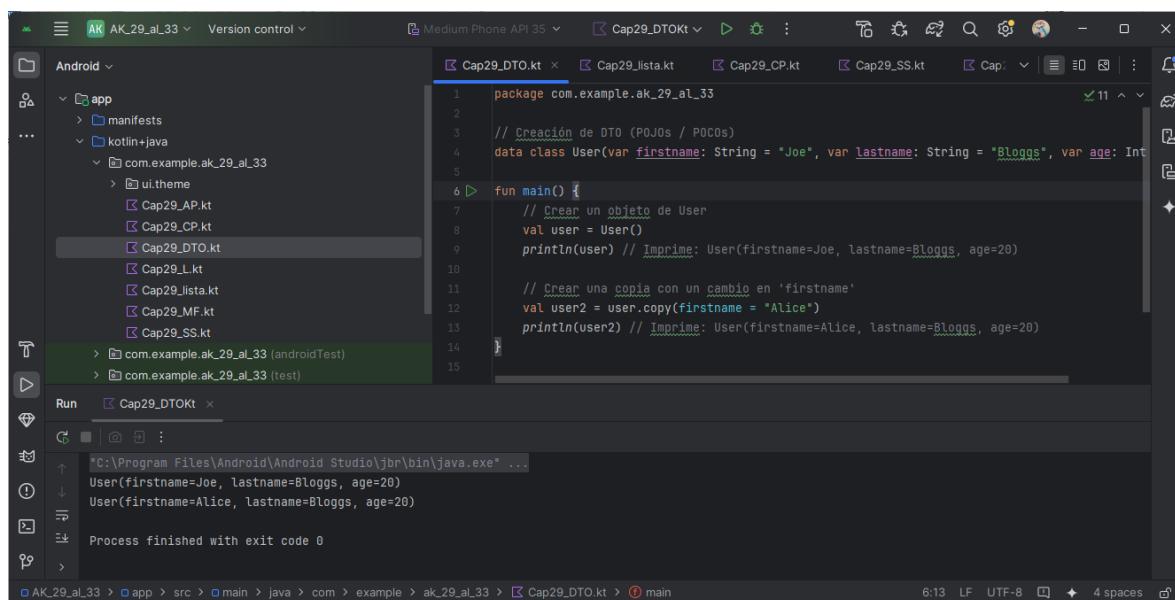


```

1 package com.example.ak_29_al_33
2
3 // Delegado a una clase sin aportarlo en el constructor público
4 interface Table<K, V, R> {
5     fun get(key: K): V
6     fun put(key: K, value: V): R
7 }
8
9 class MyTable private constructor(table: Table<Int, Int, Int>) : Table<Int, Int, Int> by table
10 companion object {
11     fun create(): MyTable = MyTable(object : Table<Int, Int, Int> {
12         override fun get(key: Int): Int = key * 2
13         override fun put(key: Int, value: Int): Int = value + 1
14     })
15 }

```

En el otro ejercicio solicitado define una clase de datos “User”, que actúa como un DTO (objeto para transferir datos). Kotlin genera automáticamente funciones como `toString()`, `equals()`, `copy()` y `componentN()` para clases data. En `main()`, se crea un usuario con valores por defecto y luego se crea una copia modificando solo el `firstname`, demostrando el uso práctico de la función `copy()`.

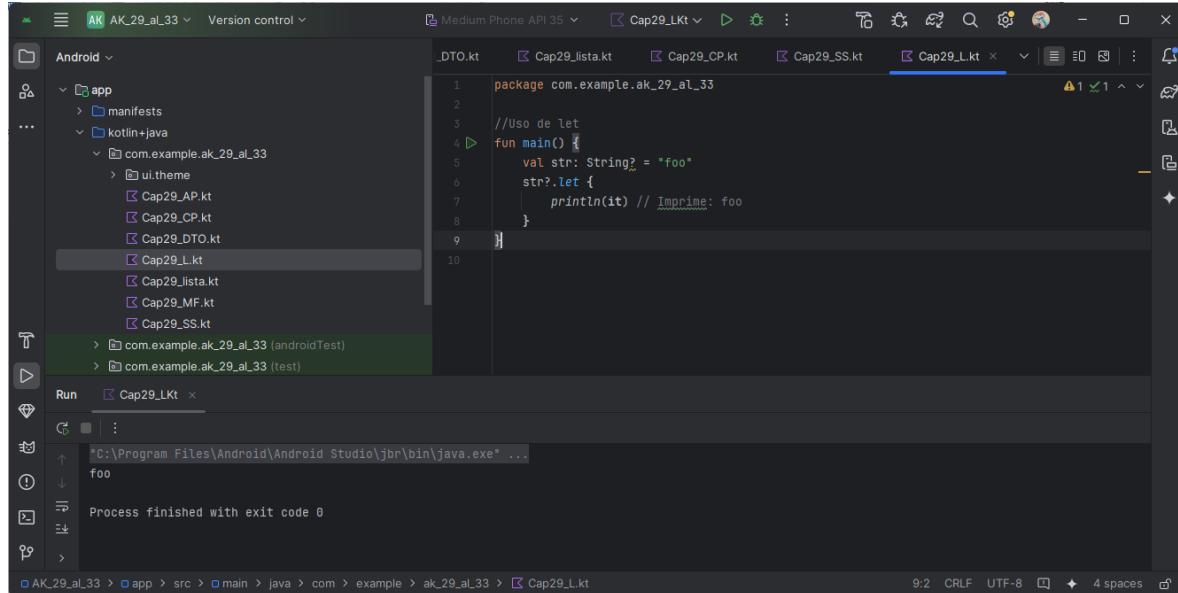


```

1 package com.example.ak_29_al_33
2
3 // Creación de DTO (POJOs / POCOs)
4 data class User(var firstname: String = "Joe", var lastname: String = "Bloggs", var age: Int = 20)
5
6 fun main() {
7     // Crear un objeto de User
8     val user = User()
9     println(user) // Imprime: User(firstname=Joe, lastname=Bloggs, age=20)
10
11     // Crear una copia con un cambio en 'firstname'
12     val user2 = user.copy(firstname = "Alice")
13     println(user2) // Imprime: User(firstname=Alice, lastname=Bloggs, age=20)
14 }
15

```

Por otra parte, en el siguiente código se demuestra el uso de “let” en Kotlin con una variable anulable. La variable str contiene “foo”, y gracias al operador seguro “?.”, el bloque let solo se ejecuta si str no es null. Dentro del bloque, it representa el valor de str, y se imprime “foo”.

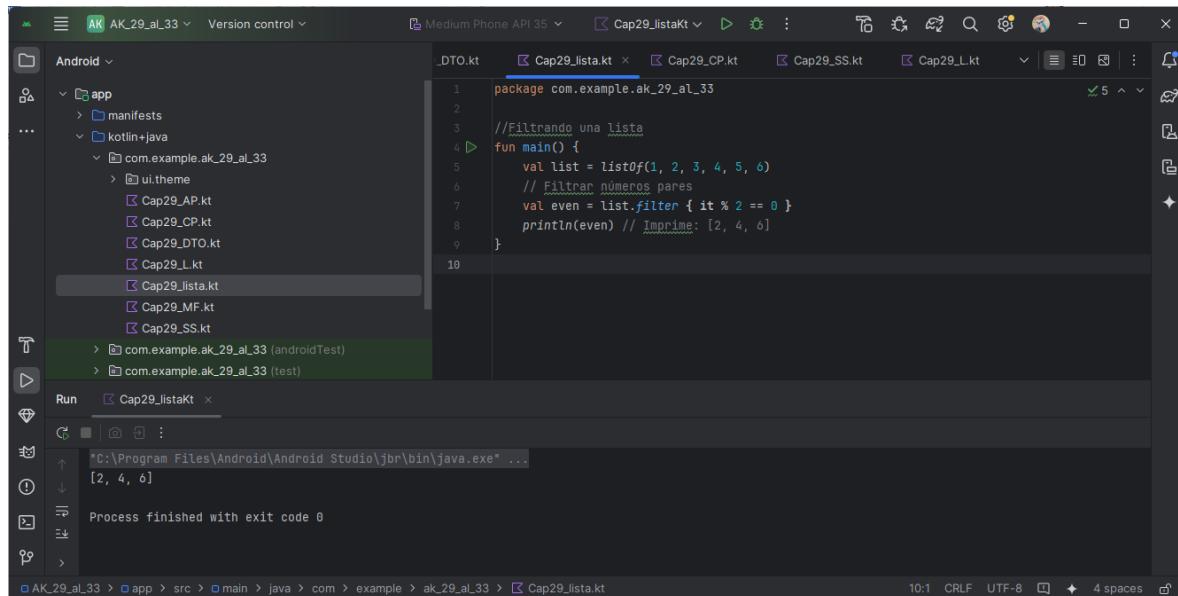


```

1 package com.example.ak_29_a1_33
2
3 //Uso de let
4 fun main() {
5     val str: String? = "foo"
6     str?.let {
7         println(it) // Imprime: foo
8     }
9 }

```

En cuanto a este otro código se muestra cómo filtrar elementos de una lista en Kotlin. Se crea una lista de números del 1 al 6 y luego se usa filter para obtener solo los números pares (it % 2 == 0). El resultado, [2, 4, 6], se imprime en consola.



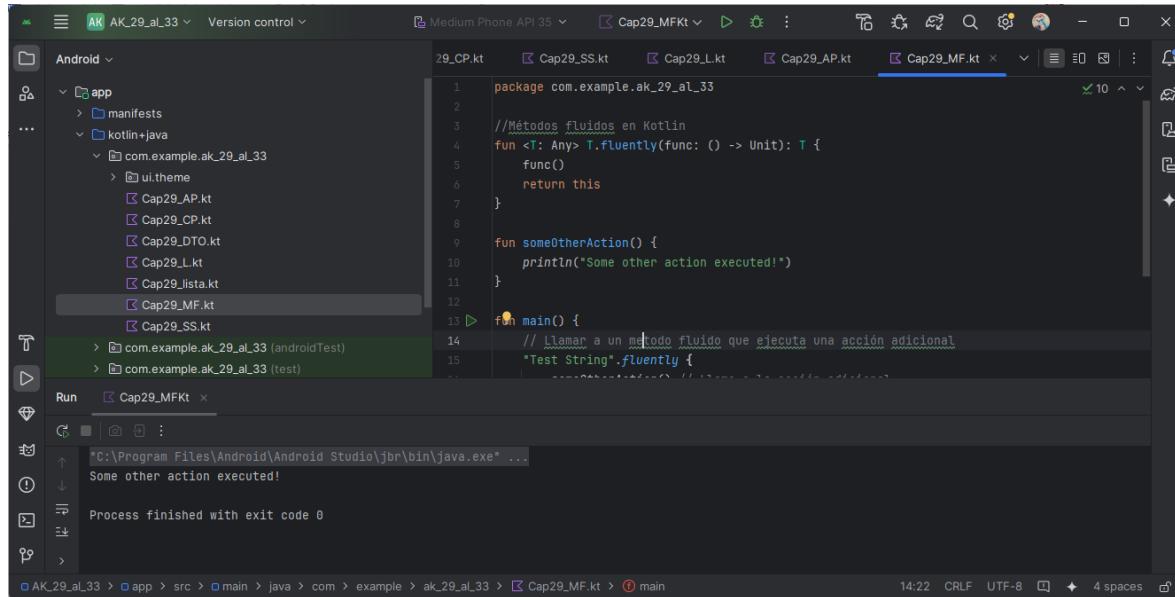
```

1 package com.example.ak_29_a1_33
2
3 //Filtrando una lista
4 fun main() {
5     val list = listOf(1, 2, 3, 4, 5, 6)
6     // Filtran números pares
7     val even = list.filter { it % 2 == 0 }
8     println(even) // Imprime: [2, 4, 6]
9 }

```

Por otra parte, en el siguiente código se muestra un ejemplo de método fluido en Kotlin usando una función de extensión. La función “fluently” ejecuta una acción (func) y luego devuelve el mismo objeto original (this). En main, se llama a fluently

sobre el string "Test String" para ejecutar someOtherAction(), que imprime un mensaje, manteniendo un estilo encadenado y claro.

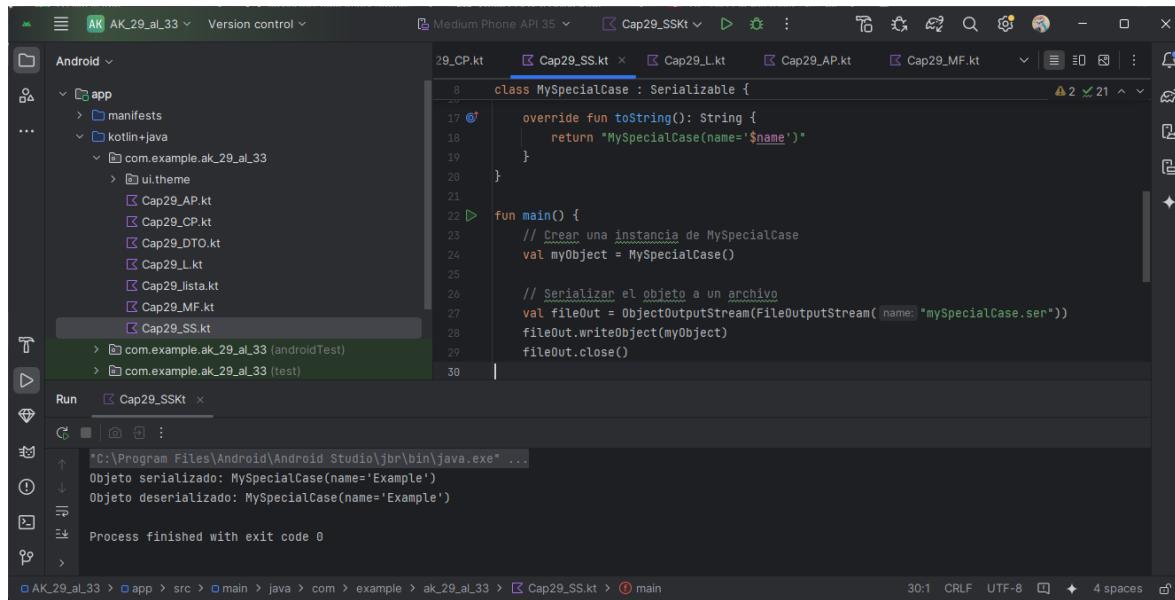


The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right. The code editor displays a file named Cap29\_MF.kt with the following content:

```
1 package com.example.ak_29_al_33
2
3 //Métodos fluidos en Kotlin
4 fun <T: Any> T.fluently(func: () -> Unit): T {
5     func()
6     return this
7 }
8
9 fun someOtherAction() {
10     println("Some other action executed!")
11 }
12
13 fun main() {
14     // Llamar a un método fluido que ejecuta una acción adicional
15     "Test String".fluently {
16         println("Another action executed!")
17     }
18 }
```

The run tab at the bottom shows the output: "Some other action executed!"

En cuanto a este otro código se muestra cómo serializar y deserializar un objeto en Kotlin. La clase "MySpecialCase" implementa "Serializable" y tiene un "serialVersionUID" para garantizar la compatibilidad de versiones. En el método main, se serializa un objeto MySpecialCase a un archivo y luego se deserializa para recuperar el objeto, imprimiendo ambos en la consola.



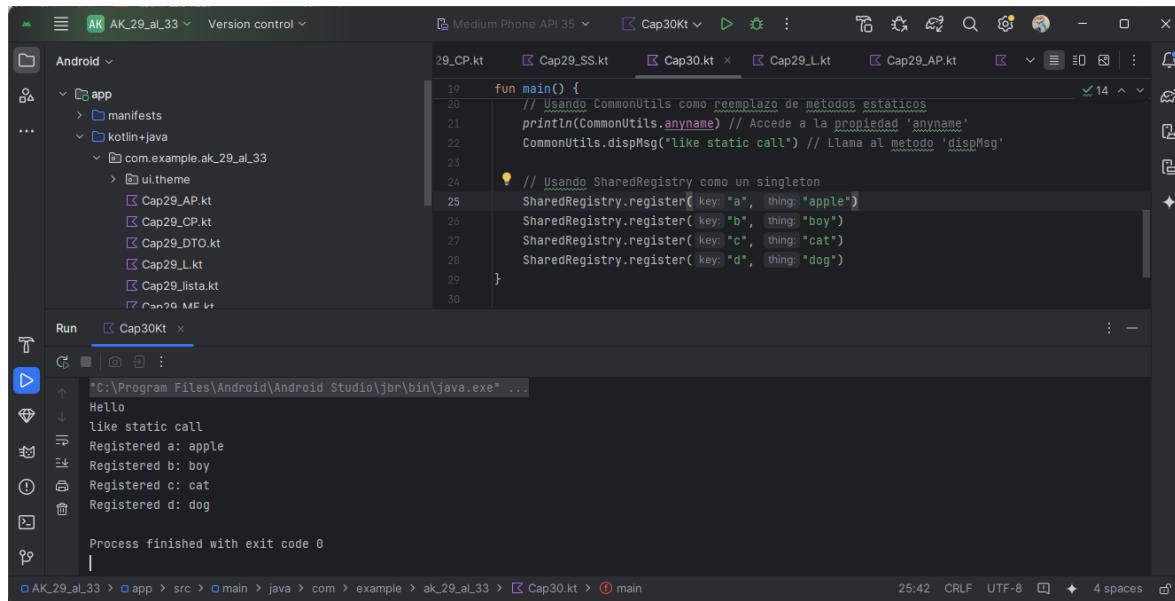
The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right. The code editor displays a file named Cap29\_SS.kt with the following content:

```
8 class MySpecialCase : Serializable {
9
10     @Override fun toString(): String {
11         return "MySpecialCase(name='$name')"
12     }
13
14     fun main() {
15         // Crear una instancia de MySpecialCase
16         val myObject = MySpecialCase()
17
18         // Serializar el objeto a un archivo
19         val fileOut = ObjectOutputStream(FileOutputStream("mySpecialCase.ser"))
20         fileOut.writeObject(myObject)
21         fileOut.close()
22     }
23 }
```

The run tab at the bottom shows the output: "Objeto serializado: MySpecialCase(name='Example') Objeto deserializado: MySpecialCase(name='Example')"

En el capítulo 30 se muestra cómo usar objetos en Kotlin para reemplazar métodos estáticos y crear singletons. "CommonUtils" simula métodos estáticos con propiedades y funciones accesibles sin crear instancias, mientras que

“SharedRegistry” actúa como un singleton, gestionando registros sin necesidad de crear un objeto explícitamente.



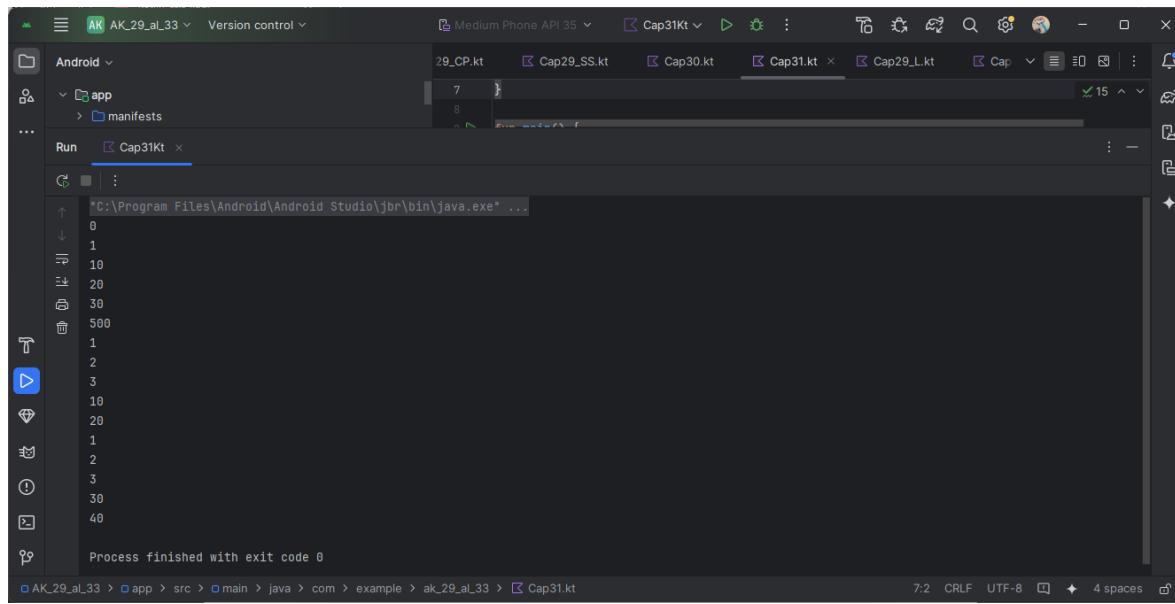
The screenshot shows the Android Studio interface with the project 'AK\_29\_aL\_33' open. The code editor displays 'Cap30Kt.kt' with the following content:

```
19 fun main() {
20     // Usando CommonUtils como reemplazo de metodos estaticos
21     println(CommonUtils.anyname) // Accede a la propiedad 'anyname'
22     CommonUtils.dispMsg("Like static call") // Llama al metodo 'dispMsg'
23
24     // Usando SharedRegistry como un singleton
25     SharedRegistry.register( key: "a", thing: "apple" )
26     SharedRegistry.register( key: "b", thing: "boy" )
27     SharedRegistry.register( key: "c", thing: "cat" )
28     SharedRegistry.register( key: "d", thing: "dog" )
29 }
```

The run tab shows the output of the program:

```
Process: C:\Program Files\Android\Android Studio\jbr\bin\java.exe ...
Hello
like static call
Registered a: apple
Registered b: boy
Registered c: cat
Registered d: dog
Process finished with exit code 0
```

En cuanto al capítulo 31 se define la función printNumbers que acepta un número variable de enteros (“vararg”) y los imprime. En main(), se muestra cómo llamar a la función con parámetros directos, pasar un array usando el operador \* y combinar ambos tipos de parámetros.



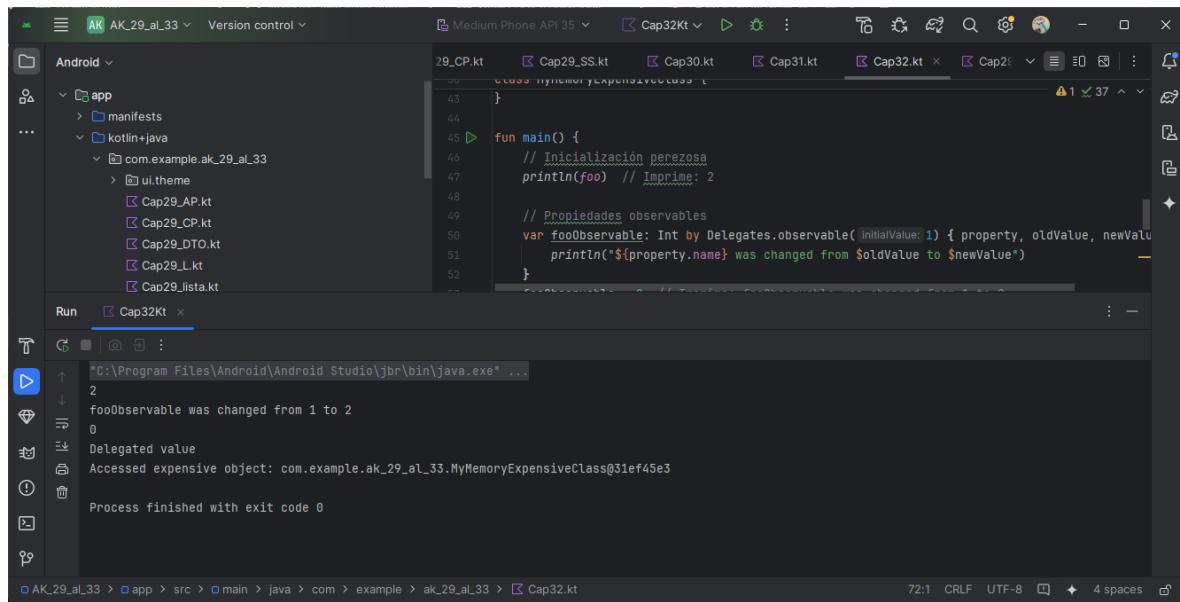
The screenshot shows the Android Studio interface with the project 'AK\_29\_aL\_33' open. The code editor displays 'Cap31Kt.kt' with the following content:

```
7
8 }
```

The run tab shows the output of the program:

```
Process: C:\Program Files\Android\Android Studio\jbr\bin\java.exe ...
0
1
10
20
30
500
1
2
3
10
20
1
2
3
30
40
Process finished with exit code 0
```

En el capítulo 32 se muestran ejemplos de delegación en Kotlin: inicialización perezosa, delegación personalizada, y uso de WeakReference para evitar fugas de memoria. También incluye propiedades observables que reaccionan a cambios de valor y mapas con valores predeterminados.

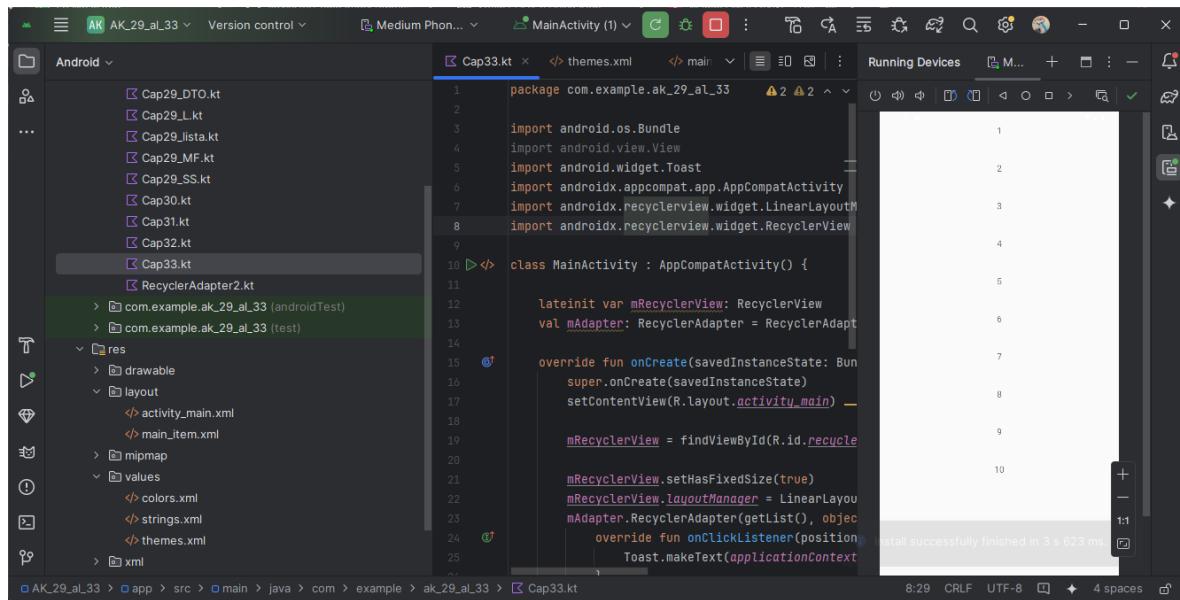


```
43
44
45 fun main() {
46     // Inicialización perezosa
47     println(foo) // Imprime: 2
48
49     // Propiedades observables
50     var fooObservable: Int by Delegates.observable(initialValue: 1) { property, oldValue, newValue
51         println("${property.name} was changed from $oldValue to $newValue")
52     }
53 }
```

Run Cap32Kt

\*C:\Program Files\Android\Android Studio\jbr\bin\java.exe\* ...  
2  
fooObservable was changed from 1 to 2  
0  
Delegated value  
Accessed expensive object: com.example.ak\_29\_al\_33.MyMemoryExpensiveClass@31ef45e3  
Process finished with exit code 0

Por último, en el capítulo 33 se utiliza un “RecyclerView” para mostrar una lista de números del 1 al 10 en una interfaz sencilla. La actividad principal (“Cap33.kt”) configura el RecyclerView con un LinearLayoutManager y un adaptador personalizado (“RecyclerAdapter2.kt”). El adaptador se encarga de crear las vistas para cada elemento de la lista y manejar los clics individuales mediante una interfaz (OnClick) que imprime en consola la posición del ítem seleccionado. El archivo “activity\_main.xml” define el diseño principal y contiene el RecyclerView, mientras que “main\_item.xml” define el diseño de cada elemento individual de la lista, que en este caso es simplemente un TextView que muestra el número. Este enfoque separa la lógica de presentación de la lógica de interacción, utilizando componentes tradicionales de Android (no Compose) para crear una lista interactiva.



```
1 package com.example.ak_29_al_33
2
3 import android.os.Bundle
4 import android.view.View
5 import android.widget.Toast
6 import androidx.appcompat.app.AppCompatActivity
7 import androidx.recyclerview.widget.LinearLayoutManager
8 import androidx.recyclerview.widget.RecyclerView
9
10 class MainActivity : AppCompatActivity() {
11
12     lateinit var mRecyclerView: RecyclerView
13     val mAdapter: RecyclerAdapter = RecyclerAdapter()
14
15     override fun onCreate(savedInstanceState: Bundle) {
16         super.onCreate(savedInstanceState)
17         setContentView(R.layout.activity_main)
18
19         mRecyclerView = findViewById(R.id.recycle)
20
21         mRecyclerView.setHasFixedSize(true)
22         mRecyclerView.setLayoutManager(LinearLayoutManager(this))
23         mAdapter.RecyclerAdapter(getList(), object : RecyclerAdapter.OnClick {
24             override fun onClickListener(position: Int) {
25                 Toast.makeText(applicationContext, "Clicked $position", Toast.LENGTH_SHORT).show()
26             }
27         })
28     }
29
30     private val getList: List<String> get() = (1..10).map { it.toString() }
31 }
```

MainActivity (I)

Running Devices

1  
2  
3  
4  
5  
6  
7  
8  
9  
10

En cuanto a los [capítulos 34, 35, 36, 37 y 38](#) en el capítulo 34 se usa reflexión para acceder y modificar propiedades de clases en tiempo de ejecución. Se muestra cómo hacer referencia a clases y funciones, y cómo obtener y modificar propiedades mutables de una clase, incluso las privadas, usando reflexión. También se emplea un delegado para gestionar el acceso a propiedades. La reflexión permite manipular el estado de las clases sin conocer sus detalles previamente.

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

- Project Structure:** Shows the project "AK\_34\_al\_38" with modules "app" and "kotlin+java". Inside "kotlin+java", there's a package "com.example.ak\_34\_al\_38" containing files "ui.theme", "Cap34.kt", and "com.example.ak\_34\_al\_38 (androidTest)".
- Main Activity:** The file "Cap34.kt" is open, displaying the following code:

```
package com.example.ak_34_al_38
import kotlin.reflect.KClass
import kotlin.reflect.KProperty
import kotlin.reflect.full.memberProperties
import kotlin.reflect.KMutableProperty
import kotlin.reflect.jvm.isAccessible
// Clase MyClass agregada
```
- Run Tab:** Shows the command "C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ... followed by class definitions and error messages related to reflection and visibility.
- Tool Window:** A tooltip is displayed over the code, explaining the `@SinceKotlin(version = "1.1")` annotation and the `KVisibility` enum. It notes that `KVisibility` does not correspond to Java's package-private and protected visibilities.
- Bottom Status Bar:** Shows "29:1 LF UTF-8 4 spaces".

En el capítulo 35 se usa un patrón de visitante para comparar una cadena con expresiones regulares. La clase “RegexWhenArgument” permite comparar una cadena con patrones de “Regex”. En el bloque when, se verifica si la cadena coincide con patrones como “c|d” o “\\d+” y se imprime un mensaje según el resultado. La clase sobrescribe el método equals() para permitir comparaciones entre instancias de RegexWhenArgument y Regex.

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

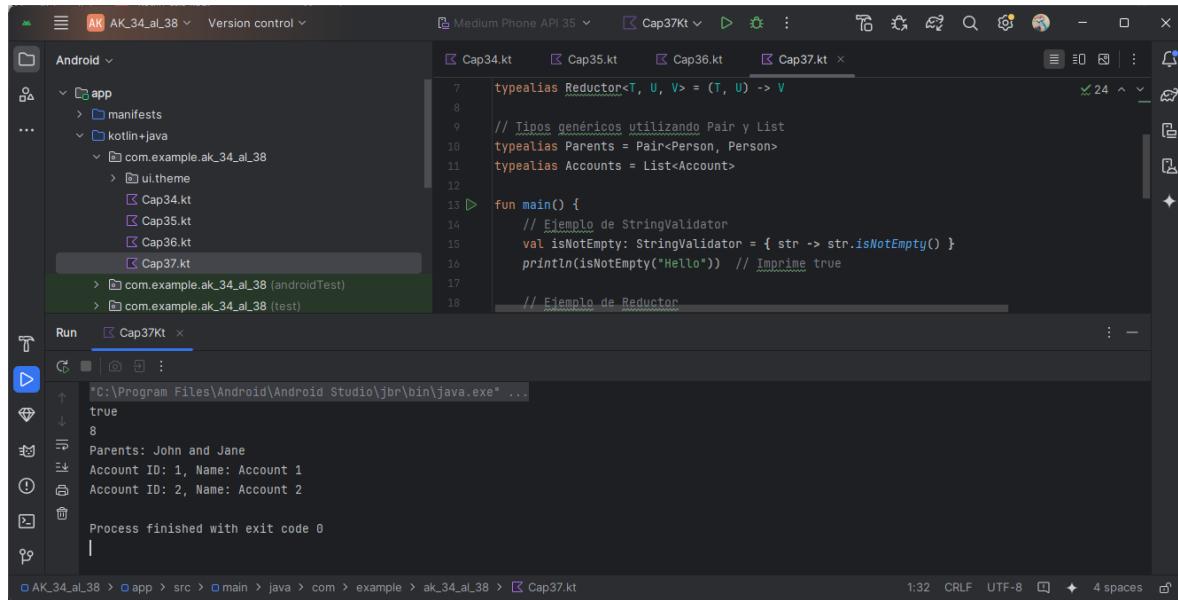
- Project Structure:** The left sidebar shows the project structure under "Android". It includes the "app" module with "manifests", "kotlin+java", and "com.example.ak\_34\_al\_38" sub-module. "Cap35.kt" is selected in the "com.example.ak\_34\_al\_38" folder.
- Code Editor:** The main editor window displays the "Cap35.kt" file. The code uses the RegexWhenArgument pattern to check if a string matches "c|d" or "\\d+".

```
fun main() {
    val string = "abc123"

    // Usando el patrón de visitante
    when (RegexWhenArgument(string)) {
        RegexWhenArgument(whenArgument: "c|d") -> println("Coincide con c o d (visitante)")
        RegexWhenArgument(whenArgument: "\\d+") -> println("Coincide con numeros (visitante)")
        else -> println("No coincide (visitante)")
    }
}
```
- Run Tab:** The "Run" tab is active, showing the command being run: "C:\Program Files\Android\Android Studio\jbr\bin\java.exe ...".
- Output:** The output pane shows the result of the run: "No coincide (visitante)".
- Bottom Status Bar:** The status bar at the bottom shows the file path as "AK\_34\_al\_38 > app > src > main > java > com > example > ak\_34\_al\_38 > Cap35.kt > RegexWhenArgument > matches", and the current time as "19:26".

En cuanto al capítulo 36 se muestra cómo manejar tipos anulables y no anulables en Kotlin. Usa operadores como “?.” para acceder a propiedades de valores anulables sin errores, “let” para operar sobre valores no nulos, y el operador Elvis (“?:”) para asignar un valor por defecto si el valor es nulo. También se filtran nulos de listas con filterNotNull() y se usa “!!” para forzar el acceso a valores no nulos, lo que puede causar excepciones si el valor es nulo.

En el capítulo 37 se usan alias de tipo en Kotlin para simplificar funciones y tipos genéricos. Define StringValidator para validar cadenas, Reductor para combinar dos tipos en uno, y Parents y Accounts como alias para Pair y List respectivamente. Los ejemplos muestran cómo usar estos alias para validar un String, sumar enteros y manejar listas de objetos Person y Account.



```
typealias Reductor<T, U, V> = (T, U) -> V
// Tipos genéricos utilizando Pair y List
typealias Parents = Pair<Person, Person>
typealias Accounts = List<Account>

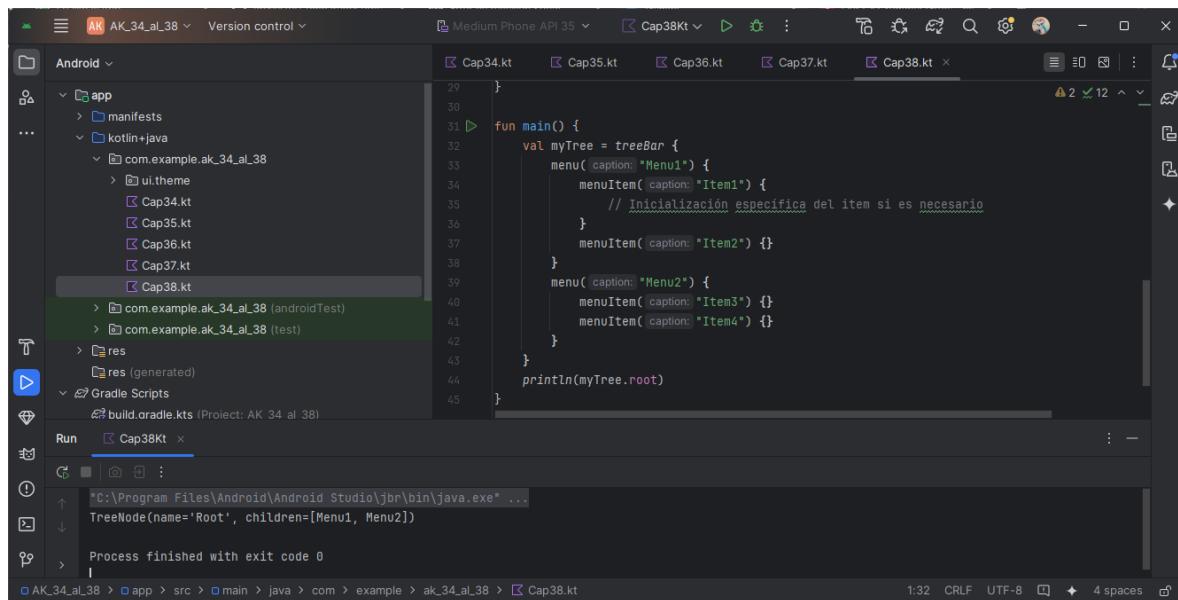
fun main() {
    // Ejemplo de StringValidator
    val isEmpty: StringValidator = { str -> str.isNotEmpty() }
    println(isEmpty("Hello")) // Imprime true

    // Ejemplo de Reductor
}
```

Output window:

```
true
8
Parents: John and Jane
Account ID: 1, Name: Account 1
Account ID: 2, Name: Account 2
Process finished with exit code 0
```

Por último, en el capítulo 38 se crea una estructura de árbol en Kotlin usando clases y funciones de extensión. La clase `TreeNode` representa un nodo con un nombre y una lista de hijos, permitiendo agregar menús y elementos de menú mediante las funciones `menu` y `menuItem`. La función `treeBar` construye un árbol con una raíz y lo llena usando lambdas. En el `main`, se construye un árbol de menús y elementos, y se imprime la estructura resultante.

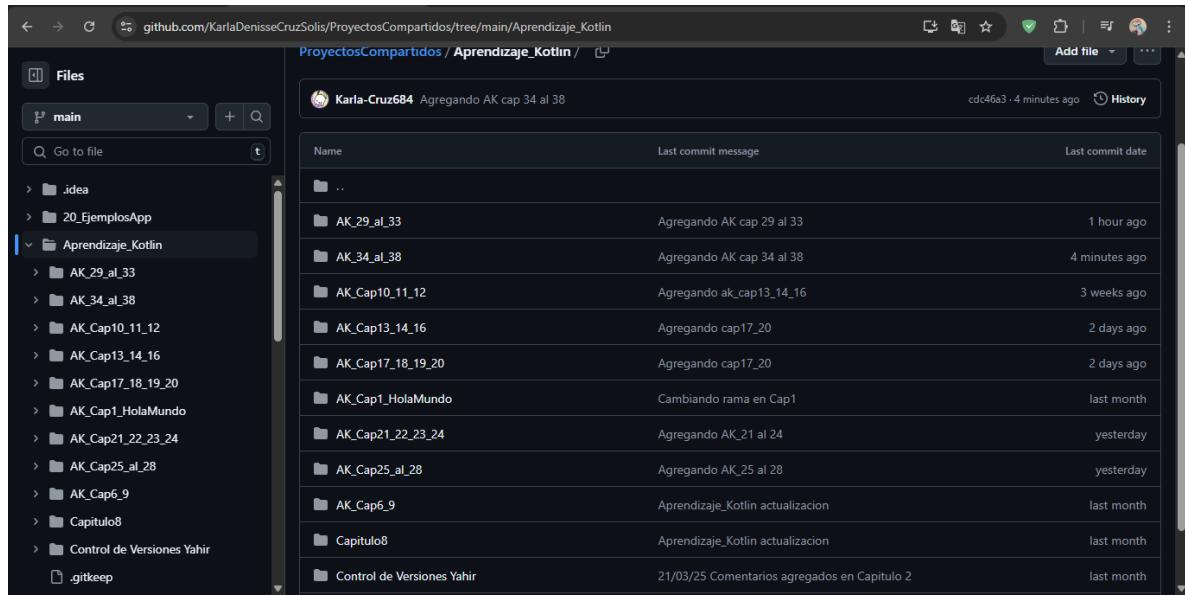


```
fun main() {
    val myTree = treeBar {
        menu( caption: "Menu1" ) {
            menuItem( caption: "Item1" ) {
                // Inicialización específica del ítem si es necesario
            }
            menuItem( caption: "Item2" ) {}
        }
        menu( caption: "Menu2" ) {
            menuItem( caption: "Item3" ) {}
            menuItem( caption: "Item4" ) {}
        }
    }
    println(myTree.root)
}
```

Output window:

```
TreeNode(name='Root', children=[Menu1, Menu2])
Process finished with exit code 0
```

Podemos ver los proyectos subidos en github de la siguiente manera:



El capítulo 1 de manera independiente, del capítulo 2 al 5 en “Control de Versiones Yahir”, del 6 al 9 donde el 8 está de manera independiente, del 10 al 12, 13 al 16, 17 al 20, 21 al 24, 25 al 28, 29 al 33, y, por último, del capítulo 34 al 38.