

Curso de Android Studio

Diego Saavedra

Sep 12, 2024

Table of contents

1	Bienvenido	4
1.1	¿De qué trata este curso?	4
1.2	¿Para quién es este curso?	4
1.3	¿Cómo contribuir?	4
I	Unidad 1: Introducción a Android	5
2	Introducción a Android Studio	6
2.1	¿Qué es Android Studio?	6
2.2	¿Por qué usar Android Studio?	6
2.3	¿Qué aprenderás en este curso?	6
2.4	¿Para quién es este curso?	7
2.5	¿Cómo contribuir?	7
2.6	¿Qué necesitas para seguir este curso?	7
2.7	¿Listo para empezar?	7
3	Instalación de Android Studio	8
3.1	Windows	8
3.2	macOS	8
3.3	Linux	9
3.4	¿Listo para empezar?	11
4	Hola Mundo en Android Studio	12
4.1	Crear un nuevo proyecto en Android Studio	12
4.2	Ejecutar el proyecto en un emulador de Android	12
4.3	Ejecutar el proyecto en un dispositivo físico	13
5	Entorno de Desarrollo de Android Studio	14
5.1	Secciones de Android Studio	14
5.1.1	Editor de Código	15
5.1.2	Barra de Herramientas	15
5.1.3	Ventana de Proyectos	16
5.1.4	Ventana de Emulador	17
5.1.5	Ventana de Terminal	17
5.2	Configuración de Android Studio.	18
6	Modo Debug en Android Studio	19
6.1	Iniciar el modo de depuración	19
6.2	Puntos de interrupción	20
6.3	Inspección de variables	21
6.4	Ejecución paso a paso	21

6.5	Finalizar el modo de depuración	22
II	Unidad 2: Lenguaje de Programación Kotlin	23
7	Kotlin el lenguaje de programación oficial para Android.	24
7.1	¿Qué es Kotlin?	24
7.2	¿Por qué Kotlin para Android?	24
7.3	¿Qué puedo programar con Kotlin?	25
7.4	¿Cómo puedo aprender Kotlin?	25
8	Hello World Kotlin	26
8.1	Crear un nuevo proyecto en Android Studio	26
8.2	Escribir el programa “Hello, World!” en Kotlin	26
9	Variables en Kotlin.	28
9.1	Declaración de variables	28
9.2	Variables mutables e inmutables	29
9.3	Inferencia de tipos	29
9.4	Ejemplo de variables en Kotlin	30
10	Tipos de Datos en Kotlin	33
10.1	Tipos de Datos Primitivos	33
10.2	Tipos de Datos de Objetos	36
10.3	Conversión de Tipos de Datos	37
11	Convenciones de codificación en Kotlin	40
11.1	Nombres de variables	40
11.2	Nombres de funciones	40
11.3	Comentarios	41
11.4	Indentación	42
11.5	Conclusiones	43
12	Comentarios en Kotlin	44
12.1	Comentarios de una sola línea	44
12.2	Comentarios de varias líneas	44
12.3	Comentarios de documentación	45
13	Sentencias if en Kotlin	46
13.1	Ejemplo de sentencia if	46
13.2	Sentencia if-else	46
13.3	Ejemplo de sentencia if-else	47
13.4	Sentencia if-else-if	47
13.5	Ejemplo de sentencia if-else-if	47
14	Sentencia when en Kotlin	51
14.1	Ejemplo de sentencia when	51

1 Bienvenido

¡Bienvenido al Curso Completo de Android Studio

Este curso está diseñado para que puedas aprender a desarrollar aplicaciones móviles con Android Studio. A lo largo de este curso aprenderás a utilizar las herramientas y técnicas necesarias para crear aplicaciones móviles de calidad profesional.

1.1 ¿De qué trata este curso?

En este curso aprenderás a desarrollar aplicaciones móviles con Android Studio. A lo largo de este curso aprenderás a utilizar las herramientas y técnicas necesarias para crear aplicaciones móviles de calidad profesional.

1.2 ¿Para quién es este curso?

Este curso está diseñado para personas que desean aprender a desarrollar aplicaciones móviles con Android Studio. No es necesario tener conocimientos previos de programación, ya que este curso está diseñado para principiantes.

1.3 ¿Cómo contribuir?

Si deseas contribuir a este curso, puedes hacerlo a través de GitHub. Puedes enviar tus sugerencias, correcciones o mejoras a través de un pull request.

¡Todas las contribuciones son bienvenidas!

Part I

Unidad 1: Introducción a Android

2 Introducción a Android Studio



Figure 2.1: Android Studio

Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones móviles en Android. Android Studio es un IDE basado en IntelliJ IDEA y es compatible con la mayoría de las plataformas de desarrollo de Android.

2.1 ¿Qué es Android Studio?

Android Studio es un entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones móviles en Android. Android Studio es un IDE basado en IntelliJ IDEA y es compatible con la mayoría de las plataformas de desarrollo de Android.

2.2 ¿Por qué usar Android Studio?

Android Studio es el IDE oficial para el desarrollo de aplicaciones móviles en Android. Android Studio ofrece una amplia gama de herramientas y características que facilitan el desarrollo de aplicaciones móviles de calidad profesional.

2.3 ¿Qué aprenderás en este curso?

En este curso aprenderás a utilizar Android Studio para desarrollar aplicaciones móviles de calidad profesional. A lo largo de este curso aprenderás a utilizar las herramientas y técnicas necesarias para crear aplicaciones móviles de calidad profesional.

2.4 ¿Para quién es este curso?

Este curso está diseñado para personas que desean aprender a desarrollar aplicaciones móviles con Android Studio. No es necesario tener conocimientos previos de programación, ya que este curso está diseñado para principiantes.

2.5 ¿Cómo contribuir?

Si deseas contribuir a este curso, puedes hacerlo a través de GitHub. Puedes enviar tus sugerencias, correcciones o mejoras a través de un pull request. ¡Todas las contribuciones son bienvenidas!

2.6 ¿Qué necesitas para seguir este curso?

Para seguir este curso necesitarás tener instalado Android Studio en tu computadora. Puedes descargar Android Studio desde el sitio web oficial de Android Studio.

2.7 ¿Listo para empezar?

¡Vamos a empezar! En la próxima lección aprenderás a instalar Android Studio en tu computadora.

3 Instalación de Android Studio



Figure 3.1: Android Studio

Para instalar Android Studio en tu computadora, depende mucho del Sistema Operativo que tengas. A continuación, te muestro cómo instalar Android Studio en los sistemas operativos más comunes.

3.1 Windows

Para instalar Android Studio en Windows, sigue los siguientes pasos:

1. Descarga Android Studio desde el sitio web oficial de Android Studio.
2. Ejecuta el instalador de Android Studio.
3. Sigue las instrucciones del instalador para instalar Android Studio en tu computadora.
4. Una vez instalado Android Studio, ábrelo y sigue las instrucciones del asistente de configuración para configurar Android Studio.

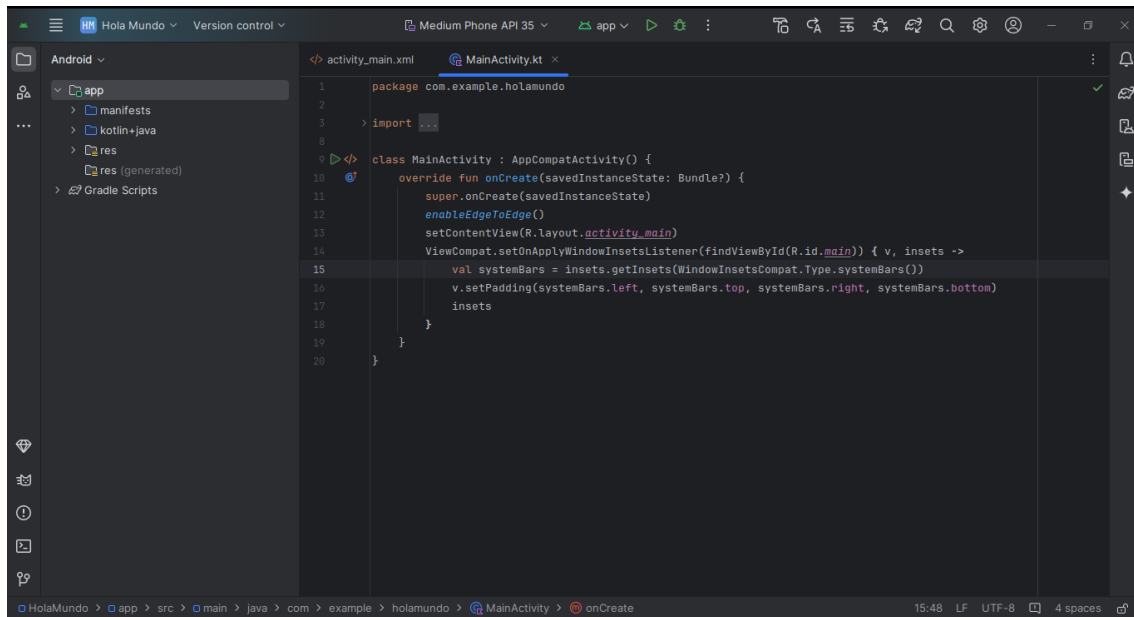
3.2 macOS

Para instalar Android Studio en macOS, sigue los siguientes pasos:

1. Descarga Android Studio desde el sitio web oficial de Android Studio.
2. Abre el archivo .dmg que descargaste.

3. Arrastra el icono de Android Studio a la carpeta de Aplicaciones.
4. Abre Android Studio desde la carpeta de Aplicaciones.
5. Sigue las instrucciones del asistente de configuración para configurar Android Studio.

3.3 Linux



Al existir distintas distribuciones de Linux, la instalación de Android Studio puede variar. A continuación, te muestro cómo instalar Android Studio en Fedora 40. Para otras distribuciones de Linux, consulta la documentación oficial de Android Studio.

Para instalar Android Studio en Linux, sigue los siguientes pasos:

1. Descarga Android Studio desde el sitio web oficial de Android Studio.
2. Extrae el archivo .zip que descargaste.

```
tar -xzf android-studio-2024.1.2.12-linux.tar.gz
```

3. Mueve el directorio extraído a /opt.

```
sudo mv android-studio /opt
```

4. Configura el entorno de Android Studio.

```
sudo ln -s /opt/android-studio/bin/studio.sh /usr/local/bin/android-studio
```

5. Iniciar Android Studio.

android-studio

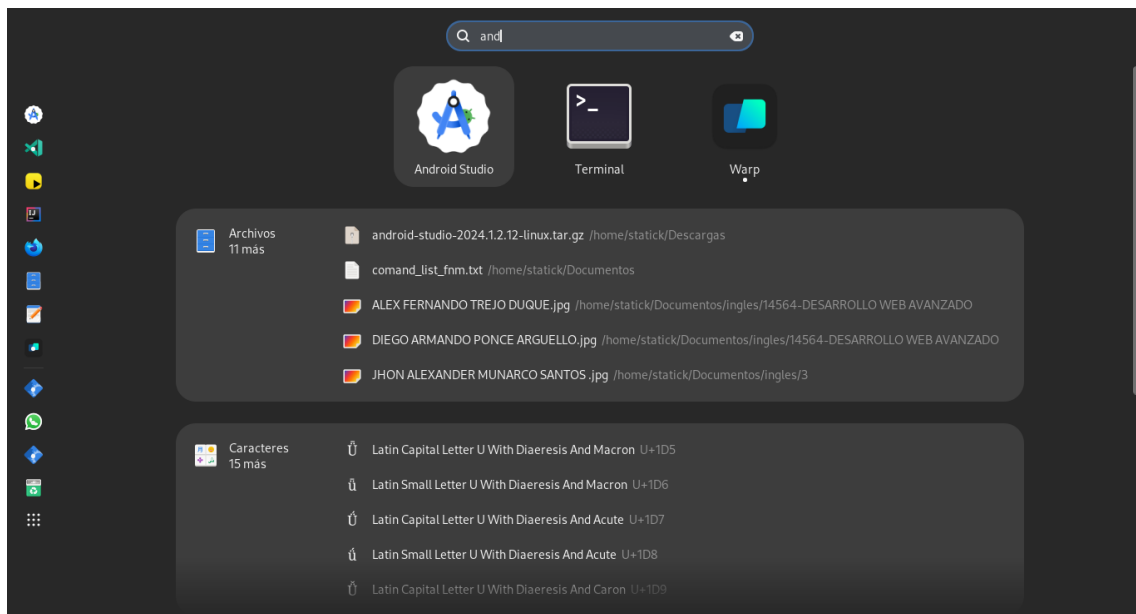
O desde el menú de aplicaciones de tu distribución Linux.

6. Configura Android Studio, La primera vez que inicies Android Studio, te pedirá que instales algunos componentes adicionales, como el Android SDK. Sigue las instrucciones en pantalla para completar la configuración inicial.
7. Opcional: Crear un acceso directo en el escritorio, puedes crear un archivo .desktop para Android Studio. Crea un archivo llamado android-studio.desktop en `~/.local/share/applications/` con el siguiente contenido:

```
[Desktop Entry]
Version=1.0
Name=Android Studio
Comment=Android Studio IDE
Exec=/opt/android-studio/bin/studio.sh
Icon=/opt/android-studio/bin/studio.png
Terminal=false
Type=Application
Categories=Development;IDE;
```

8. Guarda el archivo y actualiza el caché de aplicaciones si es necesario:

```
update-desktop-database ~/.local/share/applications
```



¡Eso es todo! Ahora deberías tener Android Studio instalado y funcionando.

3.4 ¿Listo para empezar?

¡Ahora que has instalado Android Studio en tu computadora, estás listo para empezar a desarrollar aplicaciones móviles! En la próxima lección aprenderás a crear tu primer proyecto en Android Studio.

4 Hola Mundo en Android Studio



Figure 4.1: Android Studio

En esta lección aprenderás a crear tu primer proyecto en Android Studio y a ejecutarlo en un emulador de Android.

4.1 Crear un nuevo proyecto en Android Studio

Para crear un nuevo proyecto en Android Studio, sigue los siguientes pasos:

1. Abre Android Studio en tu computadora.
2. En la pantalla de bienvenida, haz clic en “Start a new Android Studio project”.
3. Selecciona “Empty Activity” y haz clic en “Next”.
4. Completa los campos “Name” y “Package name” y haz clic en “Finish”.
5. Android Studio creará un nuevo proyecto con una actividad principal.

4.2 Ejecutar el proyecto en un emulador de Android

Para ejecutar el proyecto en un emulador de Android, sigue los siguientes pasos:

1. En Android Studio, haz clic en el botón “Run” para compilar y ejecutar el proyecto.
2. Selecciona un emulador de Android o crea uno nuevo.
3. Haz clic en “OK” para ejecutar el proyecto en el emulador.

4. El proyecto se ejecutará en el emulador y verás la pantalla de la actividad principal.

4.3 Ejecutar el proyecto en un dispositivo físico

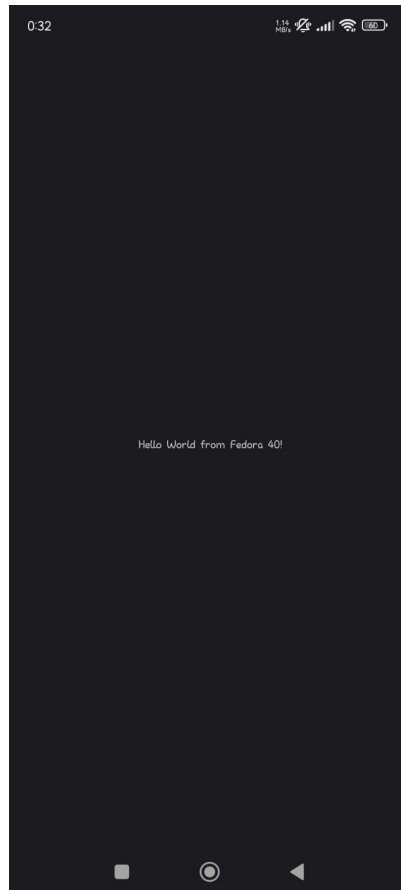


Figure 4.2: Android Device

Si deseas ejecutar el proyecto en un dispositivo físico, sigue los siguientes pasos:

1. Conecta tu dispositivo Android a tu computadora mediante un cable USB o a través de Wi-Fi.
2. En Android Studio, haz clic en el botón “Run” para compilar y ejecutar el proyecto.
3. Selecciona tu dispositivo Android en la lista de dispositivos conectados.
4. Haz clic en “OK” para ejecutar el proyecto en tu dispositivo Android.
5. El proyecto se ejecutará en tu dispositivo Android y verás la pantalla de la actividad principal.

¡Felicidades! Has creado y ejecutado tu primer proyecto en Android Studio. En la próxima lección aprenderás a personalizar la interfaz de usuario de tu aplicación.

5 Entorno de Desarrollo de Android Studio

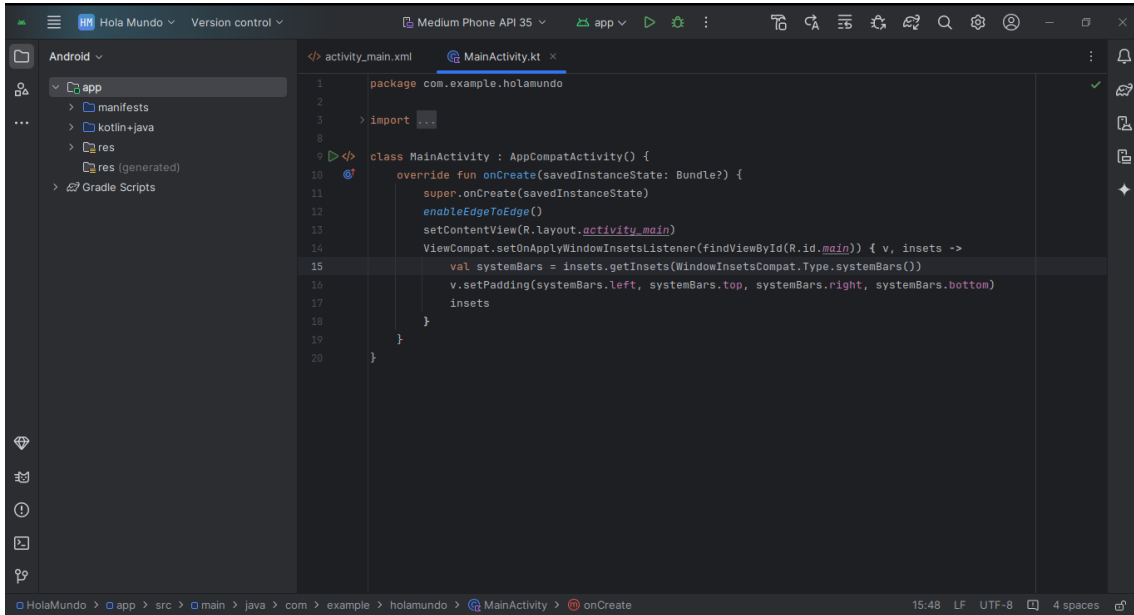


Figure 5.1: Android Studio

En esta lección aprenderás a configurar el entorno de desarrollo de Android Studio. Android Studio es un entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones móviles en Android. Android Studio es un IDE basado en IntelliJ IDEA y es compatible con la mayoría de las plataformas de desarrollo de Android.

5.1 Secciones de Android Studio

Android Studio está compuesto por varias secciones que te permiten desarrollar aplicaciones móviles de calidad profesional.

A continuación, se describen las secciones principales de Android Studio:

5.1.1 Editor de Código

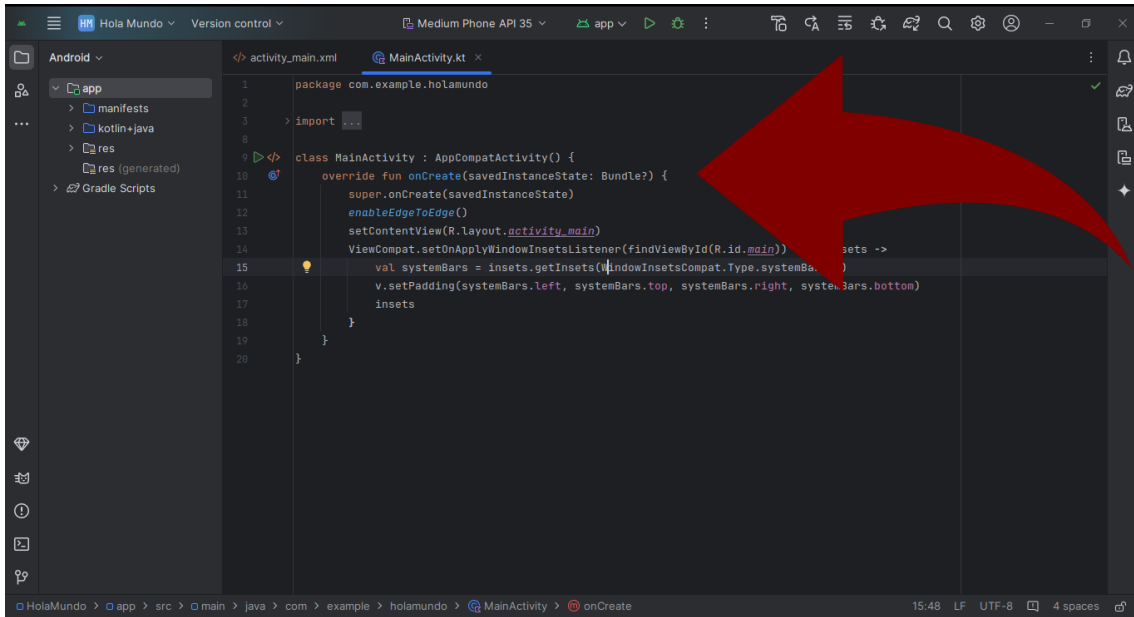


Figure 5.2: Editor de Código

El editor de código es donde escribirás el código fuente de tu aplicación. El editor de código de Android Studio es altamente personalizable y te permite escribir y editar código de forma eficiente.

5.1.2 Barra de Herramientas

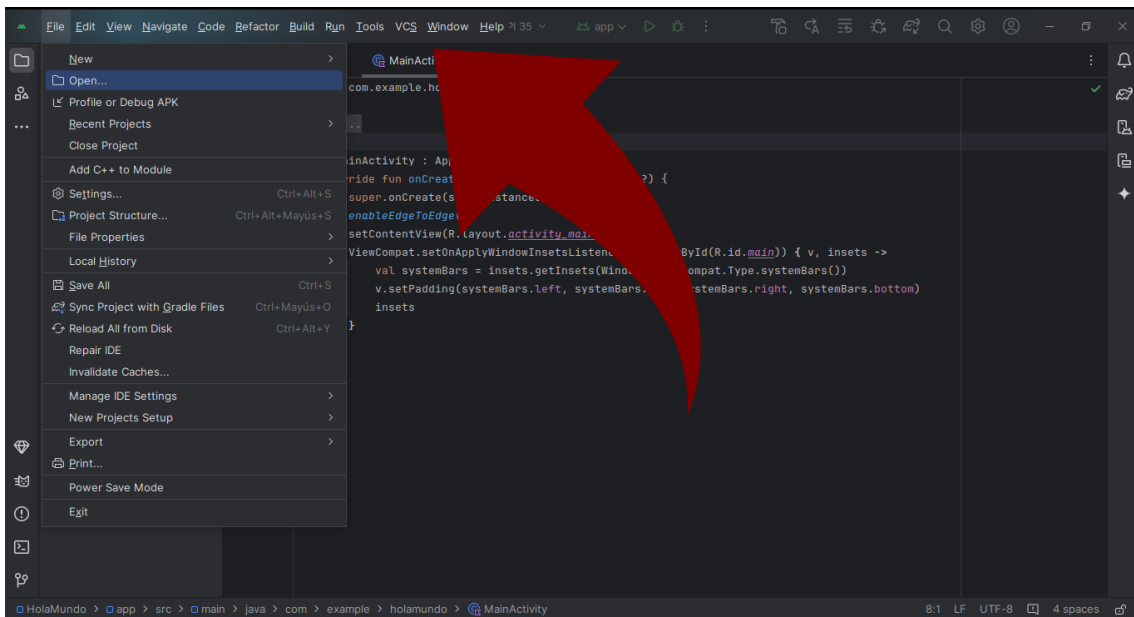
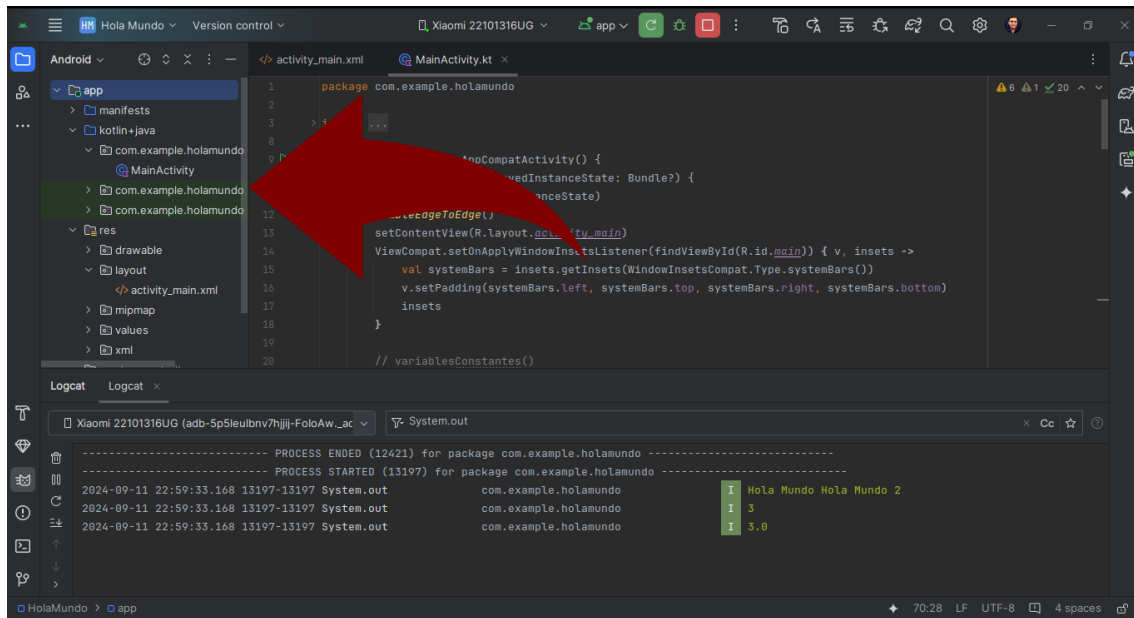


Figure 5.3: Barra de Herramientas

La barra de herramientas contiene botones y controles que te permiten compilar, ejecutar y depurar tu aplicación. La barra de herramientas también contiene botones para navegar por el código y realizar otras acciones relacionadas con el desarrollo de la aplicación.

5.1.3 Ventana de Proyectos



La ventana de proyectos muestra la estructura de tu proyecto y te permite navegar por los archivos y directorios de tu aplicación. Desde la ventana de proyectos puedes crear nuevos archivos, directorios y paquetes, y realizar otras acciones relacionadas con la gestión de tu proyecto.

5.1.4 Ventana de Emulador

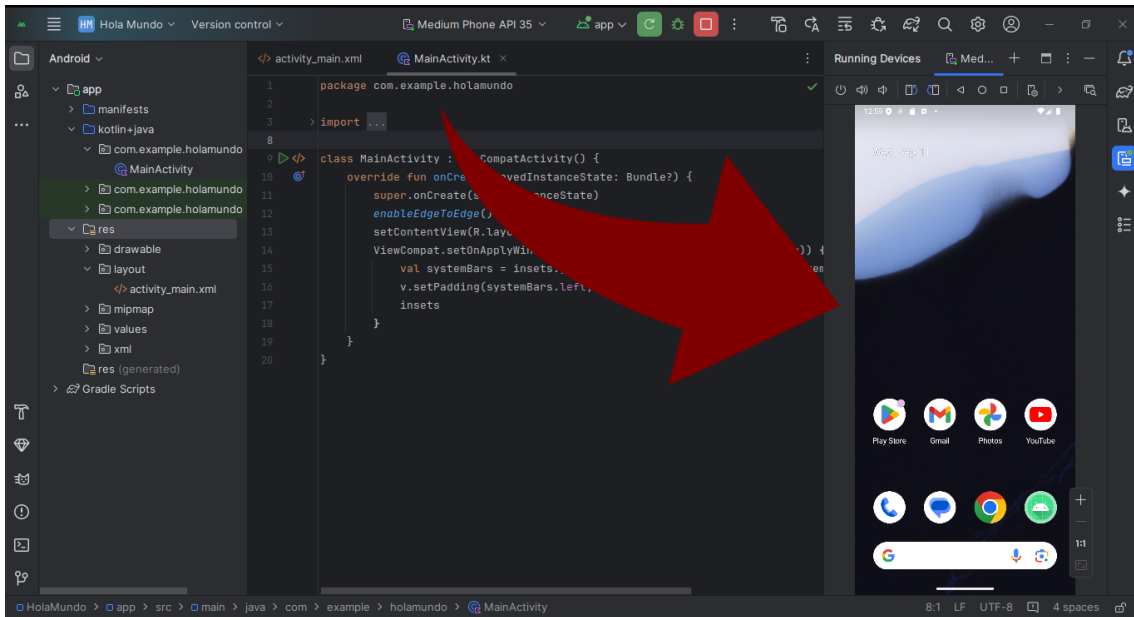


Figure 5.4: Ventana de Emulador

La ventana de emulador muestra el emulador de Android en el que puedes ejecutar y probar tu aplicación. El emulador de Android te permite probar tu aplicación en diferentes versiones de Android y tamaños de pantalla.

5.1.5 Ventana de Terminal

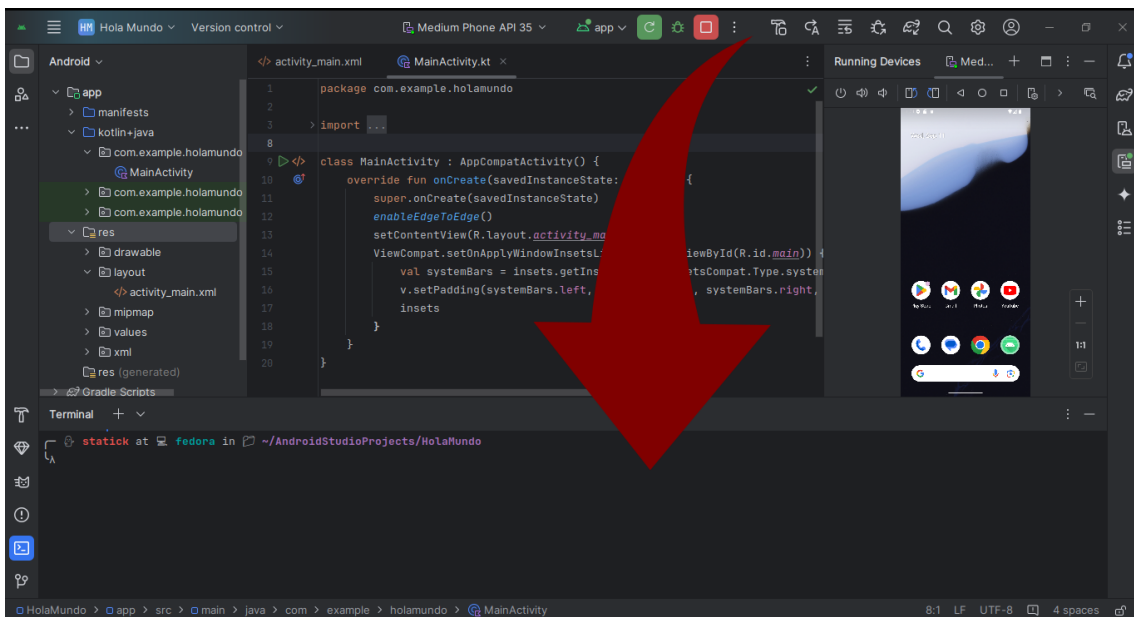
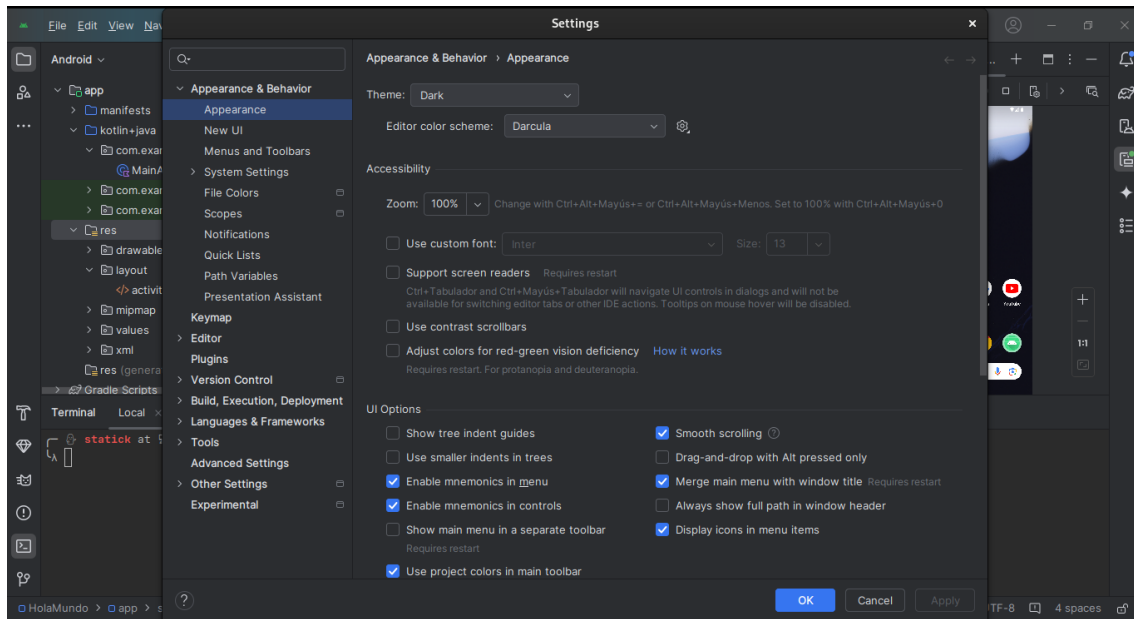


Figure 5.5: Ventana de Terminal

La ventana de terminal te permite ejecutar comandos de terminal directamente desde Android Studio. La ventana de terminal es útil para realizar tareas de desarrollo avanzadas y personalizar tu entorno de desarrollo.

5.2 Configuración de Android Studio.

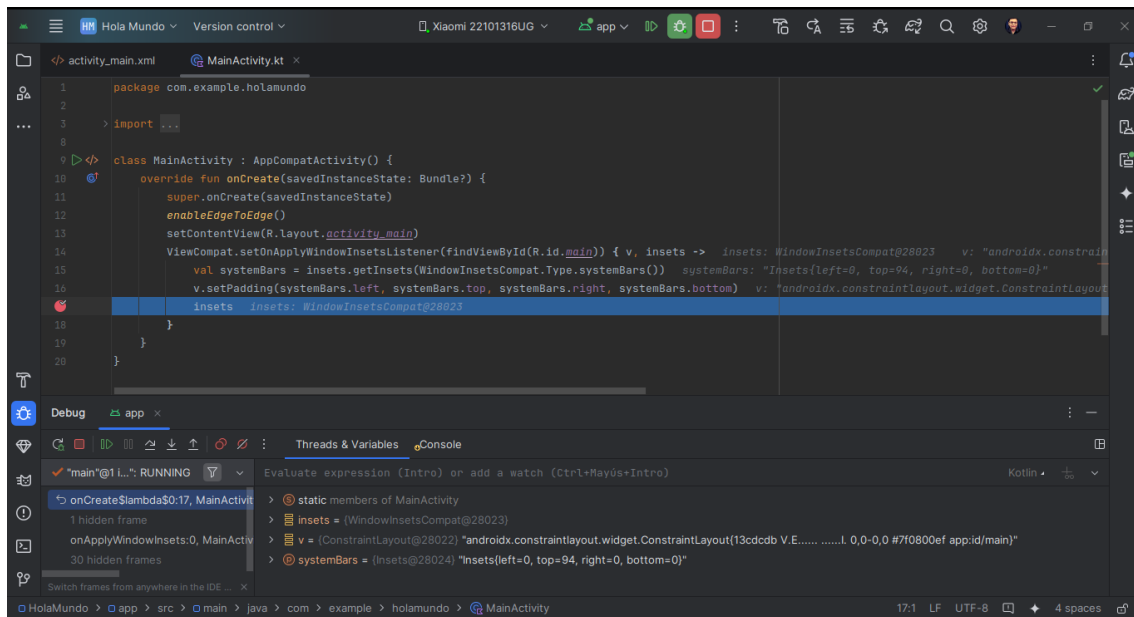


Para configurar Android Studio, sigue los siguientes pasos:

1. Abre Android Studio en tu computadora.
2. Configura las preferencias de Android Studio según tus necesidades. Puedes personalizar la apariencia, el comportamiento y las herramientas de Android
3. Importa tu proyecto existente o crea un nuevo proyecto en Android Studio.
4. Configura el SDK de Android y las herramientas de desarrollo necesarias para tu proyecto.
5. Configura el emulador de Android o conecta un dispositivo físico para probar tu aplicación.
6. Comienza a desarrollar tu aplicación en Android Studio.

¡Eso es todo! Ahora estás listo para empezar a desarrollar aplicaciones móviles con Android Studio. En la próxima lección aprenderás a crear tu primer proyecto en Android Studio.

6 Modo Debug en Android Studio



El modo de depuración (debug) en Android Studio es una característica que te permite ejecutar tu aplicación paso a paso y ver el estado de las variables y objetos en tiempo real. El modo de depuración es una herramienta poderosa que te permite identificar y corregir errores en tu aplicación de manera eficiente.

6.1 Iniciar el modo de depuración

Para iniciar el modo de depuración en Android Studio, sigue los siguientes pasos:

1. Abre tu proyecto en Android Studio.
2. Haz clic en el botón “Run” para compilar y ejecutar tu aplicación.
3. Selecciona el dispositivo en el que deseas ejecutar la aplicación.
4. Haz clic en el botón “Debug” para iniciar el modo de depuración.
5. La aplicación se ejecutará en modo de depuración y se detendrá en el primer punto de interrupción.

6.2 Puntos de interrupción

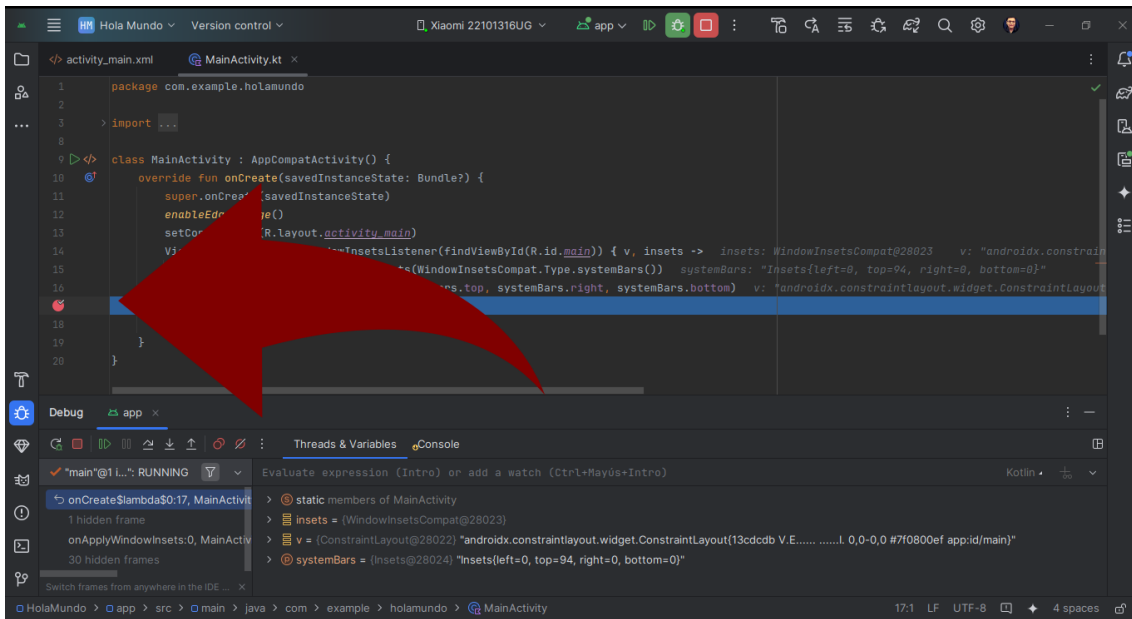


Figure 6.1: Punto de Interrupción

Los puntos de interrupción son marcadores que puedes colocar en tu código para detener la ejecución de la aplicación en un punto específico. Los puntos de interrupción te permiten inspeccionar el estado de las variables y objetos en ese punto y depurar posibles errores.

Para colocar un punto de interrupción en Android Studio, haz clic en la barra lateral izquierda del editor de código en la línea donde deseas colocar el punto de interrupción.

6.3 Inspección de variables

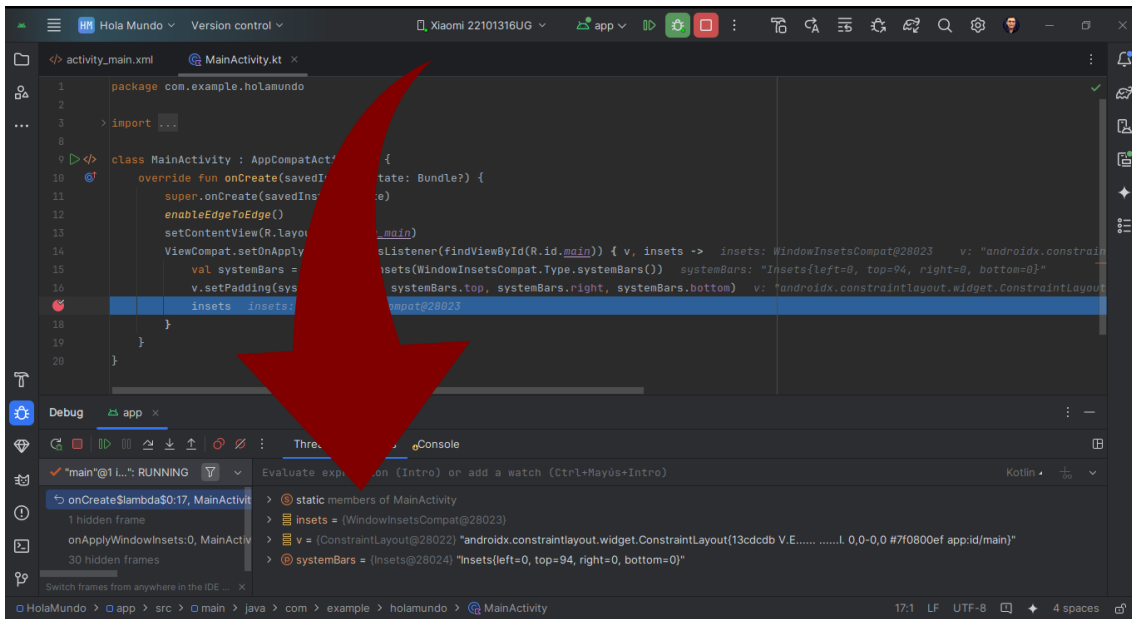


Figure 6.2: Inspección de Variables

Durante la ejecución en modo de depuración, puedes inspeccionar el estado de las variables y objetos en tiempo real. Para inspeccionar una variable, coloca el cursor sobre la variable en el editor de código y verás su valor actual.

6.4 Ejecución paso a paso

En el modo de depuración, puedes ejecutar tu aplicación paso a paso para identificar posibles errores. Puedes ejecutar la aplicación paso a paso utilizando los botones de control en la barra de herramientas de Android Studio.

6.5 Finalizar el modo de depuración

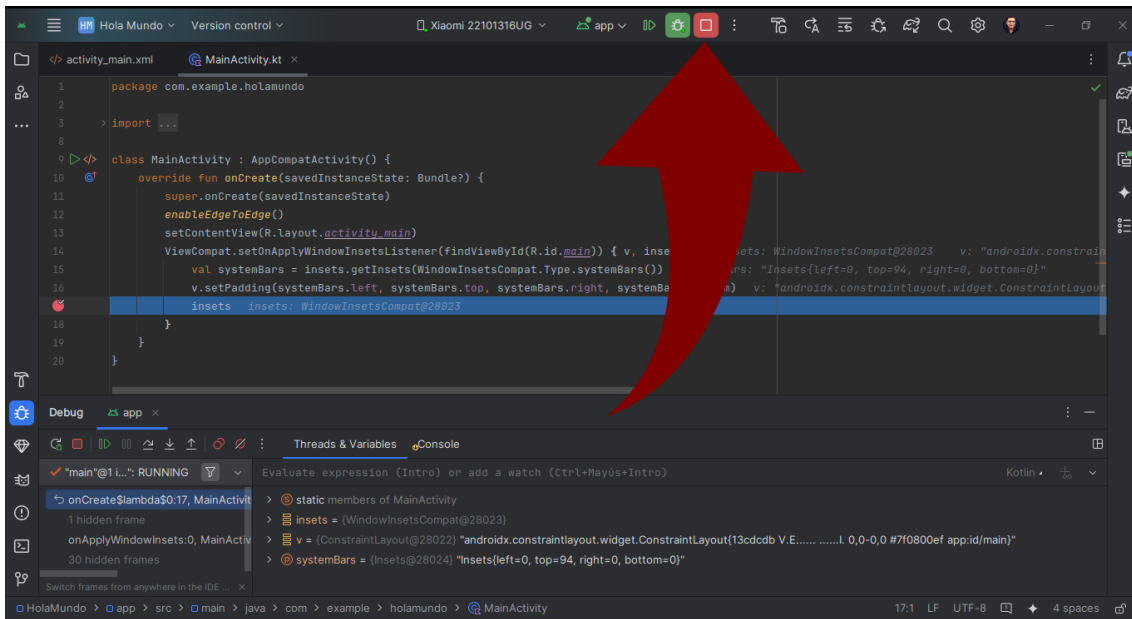


Figure 6.3: Finalizar Depuración

Para finalizar el modo de depuración en Android Studio, haz clic en el botón “Stop” en la barra de herramientas. La aplicación se detendrá y podrás continuar trabajando en tu código.

¡Felicidades! Ahora sabes cómo utilizar el modo de depuración en Android Studio para identificar y corregir errores en tu aplicación.

Part II

Unidad 2: Lenguaje de Programación Kotlin

7 Kotlin el lenguaje de programación oficial para Android.



Para aprender a programar aplicaciones Android, es necesario conocer el lenguaje de programación Kotlin. Kotlin es un lenguaje moderno, conciso y seguro que se ha convertido en el lenguaje de programación oficial para el desarrollo de aplicaciones Android.

7.1 ¿Qué es Kotlin?

Kotlin es un lenguaje de programación de tipado estático que se ejecuta en la máquina virtual de Java (JVM). Fue desarrollado por JetBrains en 2011 y se lanzó oficialmente en 2016. Kotlin está diseñado para ser interoperable con Java, lo que significa que puedes utilizar bibliotecas y frameworks de Java en tus aplicaciones Kotlin.

Kotlin es un lenguaje moderno que incorpora características avanzadas como la inferencia de tipos, las funciones de extensión, las expresiones lambda y la programación orientada a objetos. Kotlin es un lenguaje conciso que te permite escribir menos código y ser más productivo.

7.2 ¿Por qué Kotlin para Android?

Kotlin se ha convertido en el lenguaje de programación oficial para el desarrollo de aplicaciones Android por varias razones:

1. **Interoperabilidad con Java:** Kotlin es interoperable con Java, lo que significa que puedes utilizar bibliotecas y frameworks de Java en tus aplicaciones Kotlin. Esto facilita la migración de aplicaciones existentes de Java a Kotlin.

2. **Seguridad:** Kotlin es un lenguaje seguro que elimina clases de errores comunes en Java, como las excepciones de puntero nulo (`NullPointerException`). Kotlin te obliga a manejar explícitamente los valores nulos, lo que reduce la posibilidad de errores en tiempo de ejecución.
3. **Productividad:** Kotlin es un lenguaje conciso que te permite escribir menos código y ser más productivo. Kotlin incorpora características avanzadas como la inferencia de tipos, las funciones de extensión y las expresiones lambda que te permiten escribir código de manera más eficiente.
4. **Soporte oficial:** Google ha adoptado Kotlin como el lenguaje de programación oficial para el desarrollo de aplicaciones Android. Google proporciona soporte oficial para Kotlin en Android Studio, lo que facilita la creación de aplicaciones Android con Kotlin.

7.3 ¿Qué puedo programar con Kotlin?

Con Kotlin, puedes programar una amplia variedad de aplicaciones Android, incluyendo:

- Aplicaciones de productividad
- Aplicaciones de redes sociales
- Aplicaciones de mensajería
- Aplicaciones de comercio electrónico
- Juegos móviles
- Aplicaciones de mapas y geolocalización
- Aplicaciones de multimedia
- Aplicaciones de IoT (Internet de las cosas)

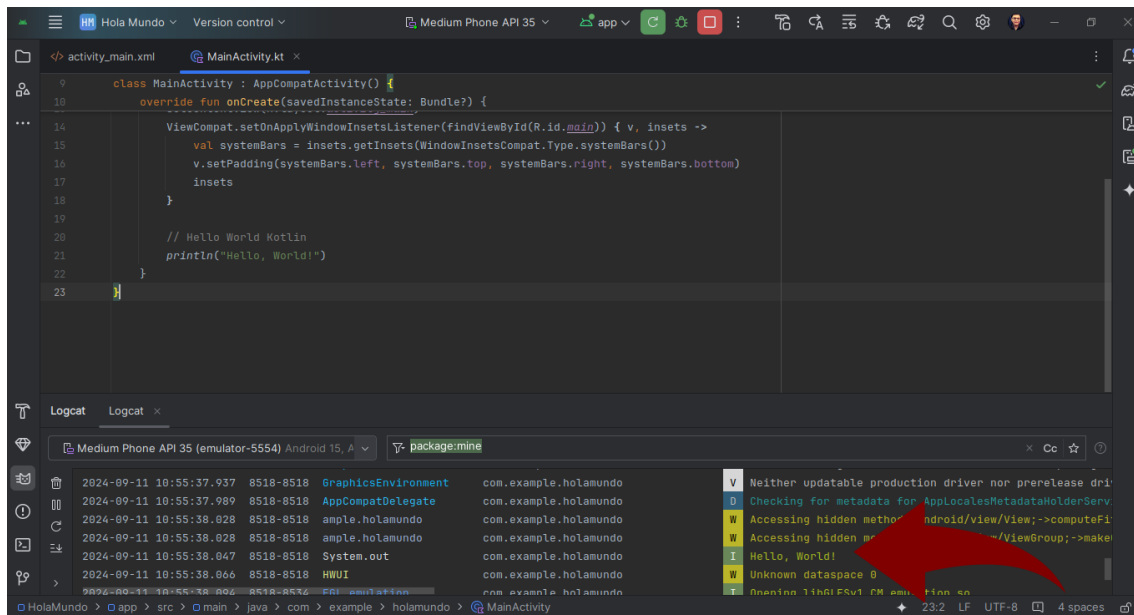
Kotlin es un lenguaje versátil que te permite programar aplicaciones Android de cualquier tipo y complejidad. Con Kotlin, puedes aprovechar las características avanzadas del lenguaje y la plataforma Android para crear aplicaciones modernas y eficientes.

7.4 ¿Cómo puedo aprender Kotlin?

Para aprender Kotlin, puedes utilizar recursos en línea como tutoriales, documentación oficial y cursos en línea. También puedes practicar programando aplicaciones Android con Kotlin y participar en comunidades de desarrolladores para compartir conocimientos y experiencias.

¡Empieza a aprender Kotlin hoy y conviértete en un desarrollador de aplicaciones Android experto!

8 Hello World Kotlin



```
9 class MainActivity : AppCompatActivity() {
10     override fun onCreate(savedInstanceState: Bundle?) {
11         // ...
12         ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets ->
13             val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
14             v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)
15             insets
16         }
17         // Hello World Kotlin
18         println("Hello, World!")
19     }
20 }
21 
```

Logcat

Time	Process	Class	Message
2024-09-11 10:55:37.937	8518-8518	GraphicsEnvironment	Neither updatable production driver nor prerelease dri
2024-09-11 10:55:37.989	8518-8518	AppCompatActivity	Checking for metadata for AppLocalMetadataHolderServ
2024-09-11 10:55:38.028	8518-8518	ample.holamundo	Accessing hidden method android.view.View;->computeFi
2024-09-11 10:55:38.028	8518-8518	ample.holamundo	Accessing hidden method android.view.ViewGroup;->make
2024-09-11 10:55:38.047	8518-8518	System.out	Hello, World!
2024-09-11 10:55:38.066	8518-8518	HWUI	Unknown datasource 0

Para crear nuestro primer programa en Kotlin, vamos a crear un proyecto nuevo en Android Studio y escribir un programa que imprima “Hello, World!” en la consola.

8.1 Crear un nuevo proyecto en Android Studio

Para crear un nuevo proyecto en Android Studio, sigue los siguientes pasos:

1. Abre Android Studio.
2. Haz clic en “Start a new Android Studio project”.
3. Selecciona “Empty Activity” y haz clic en “Next”.
4. Ingresa el nombre de tu proyecto y haz clic en “Finish”.

8.2 Escribir el programa “Hello, World!” en Kotlin

Para escribir el programa “Hello, World!” en Kotlin, sigue los siguientes pasos:

1. Abre el archivo **MainActivity.kt** en la carpeta **kotlin+java** de tu proyecto.
2. Escribe el siguiente código en el archivo **MainActivity.kt**:

```

package com.example.holamundo

import android.os.Bundle
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat

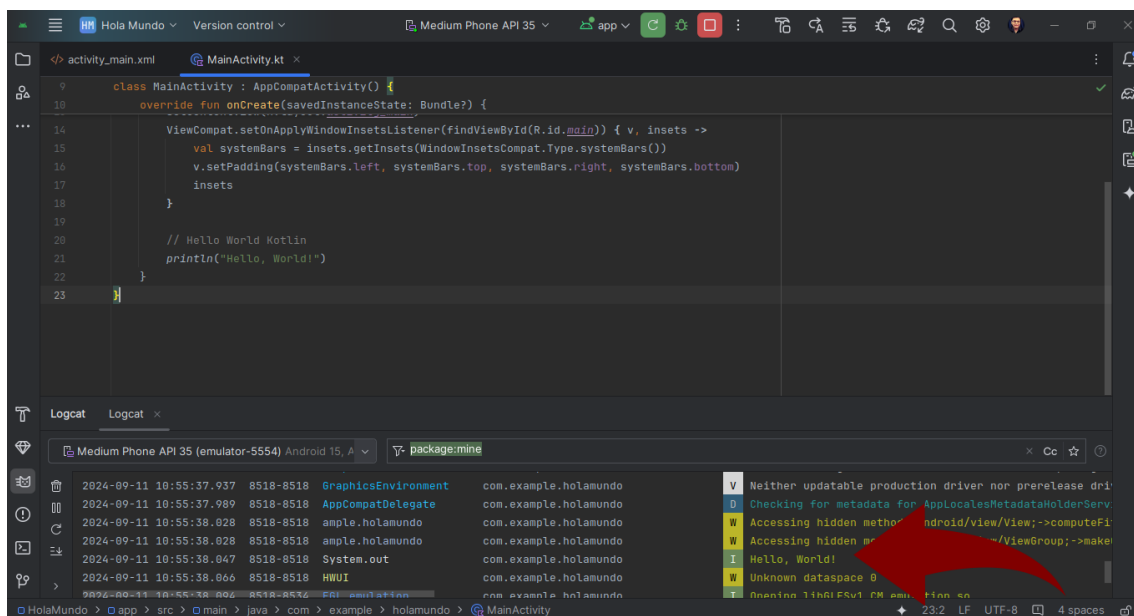
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_main)
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets ->
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)
            insets
        }

        // Hello World Kotlin
        println("Hello, World!")
    }
}

```

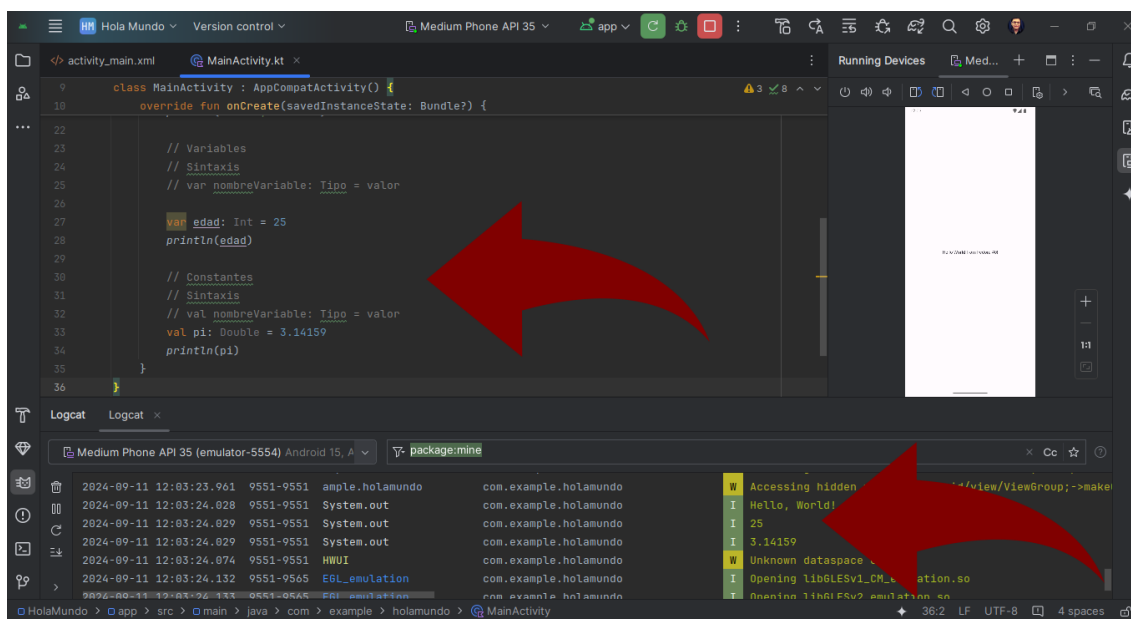
3. Haz clic en “Run” para compilar y ejecutar tu aplicación.

4. Verás “Hello, World!” impreso en la consola Logcat de Android Studio.



¡Felicidades! Has creado tu primer programa en Kotlin que imprime “Hello, World!” en la consola. Ahora puedes continuar aprendiendo Kotlin y desarrollando aplicaciones Android con Android Studio.

9 Variables en Kotlin.



Las variables son elementos fundamentales en cualquier lenguaje de programación, ya que nos permiten almacenar y manipular datos en la memoria de la computadora. En Kotlin, las variables se definen utilizando la palabra clave **var** para variables mutables y **val** para variables inmutables.

9.1 Declaración de variables

Para declarar una variable en Kotlin, se utiliza la siguiente sintaxis:

```
var nombreVariable: TipoDato = valorInicial
```

Donde:

- **nombreVariable**: Es el nombre de la variable.
- **TipoDato**: Es el tipo de dato que almacenará la variable.
- **valorInicial**: Es el valor inicial de la variable.

Por ejemplo, para declarar una variable entera llamada **edad** con un valor inicial de **25**, se utiliza la siguiente sintaxis:

```
var edad: Int = 25
```

En este caso, la variable **edad** es de tipo **Int** (entero) y tiene un valor inicial de **25**.

9.2 Variables mutables e inmutables

En Kotlin, las variables se pueden declarar como mutables utilizando la palabra clave **var** o inmutables utilizando la palabra clave **val**. Las variables mutables pueden cambiar su valor a lo largo del tiempo, mientras que las variables inmutables no pueden cambiar su valor una vez que se les asigna un valor inicial.

Por ejemplo, para declarar una variable mutable llamada **nombre** con un valor inicial de “**Juan**”, se utiliza la siguiente sintaxis:

```
var nombre: String = "Juan"
```

En este caso, la variable **nombre** es mutable y puede cambiar su valor en cualquier momento.

Por otro lado, para declarar una variable inmutable llamada **pi** con un valor inicial de **3.1416**, se utiliza la siguiente sintaxis:

```
val pi: Double = 3.1416
```

En este caso, la variable **pi** es inmutable y no puede cambiar su valor una vez que se le asigna el valor inicial.

9.3 Inferencia de tipos

En Kotlin, el compilador puede inferir el tipo de dato de una variable en función del valor inicial asignado a la variable. Esto se conoce como inferencia de tipos y permite escribir código más conciso y legible.

Por ejemplo, en lugar de declarar explícitamente el tipo de dato de una variable como en el siguiente ejemplo:

```
var nombre: String = "Juan"
```

Se puede utilizar la inferencia de tipos para declarar la variable de la siguiente manera:

```
var nombre = "Juan"
```

En este caso, el compilador infiere que la variable **nombre** es de tipo **String** en función del valor inicial “**Juan**”.

9.4 Ejemplo de variables en Kotlin

A continuación, se muestra un ejemplo de declaración de variables en Kotlin:

```
fun main() {
    // Variables mutables
    var nombre: String = "Juan"
    var edad: Int = 25
    var altura: Double = 1.75

    // Variables inmutables
    val pi: Double = 3.1416
    val gravedad: Double = 9.81

    // Inferencia de tipos
    var ciudad = "Lima"
    var temperatura = 25.5

    // Imprimir variables
    println("Nombre: $nombre")
    println("Edad: $edad años")
    println("Altura: $altura metros")
    println("Valor de Pi: $pi")
    println("Aceleración de la gravedad: $gravedad m/s2")
    println("Ciudad: $ciudad")
    println("Temperatura: $temperatura °C")
}
```

En este ejemplo, se declaran variables mutables e inmutables de diferentes tipos de datos y se utiliza la inferencia de tipos para declarar variables sin especificar el tipo de dato de forma explícita. Finalmente, se imprimen los valores de las variables en la consola.

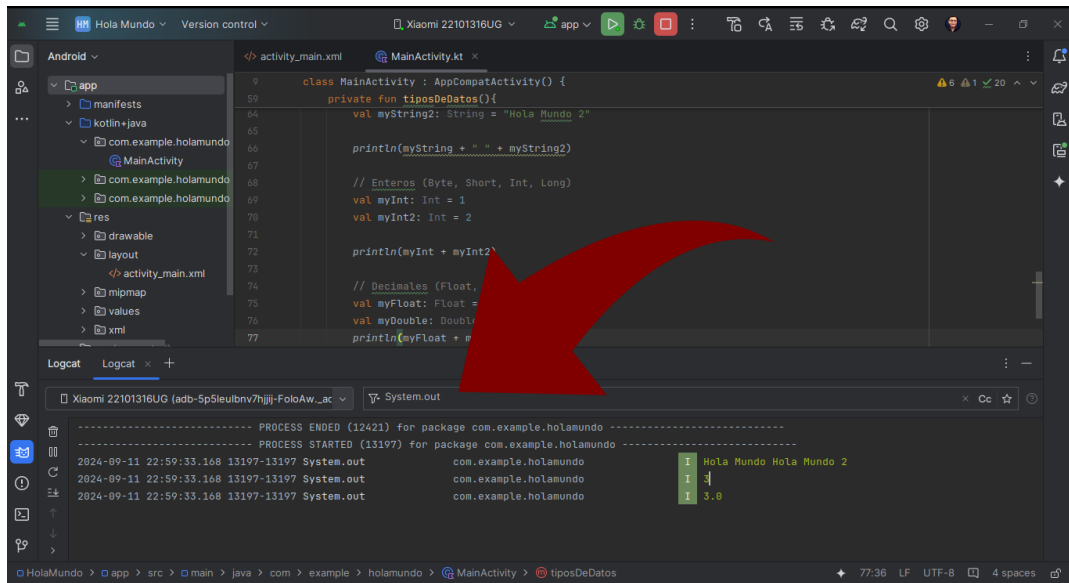
Tip

Para ver la salida solo de lo que estamos programando en la consola, podemos agregar el filtro:

```
System.out
```

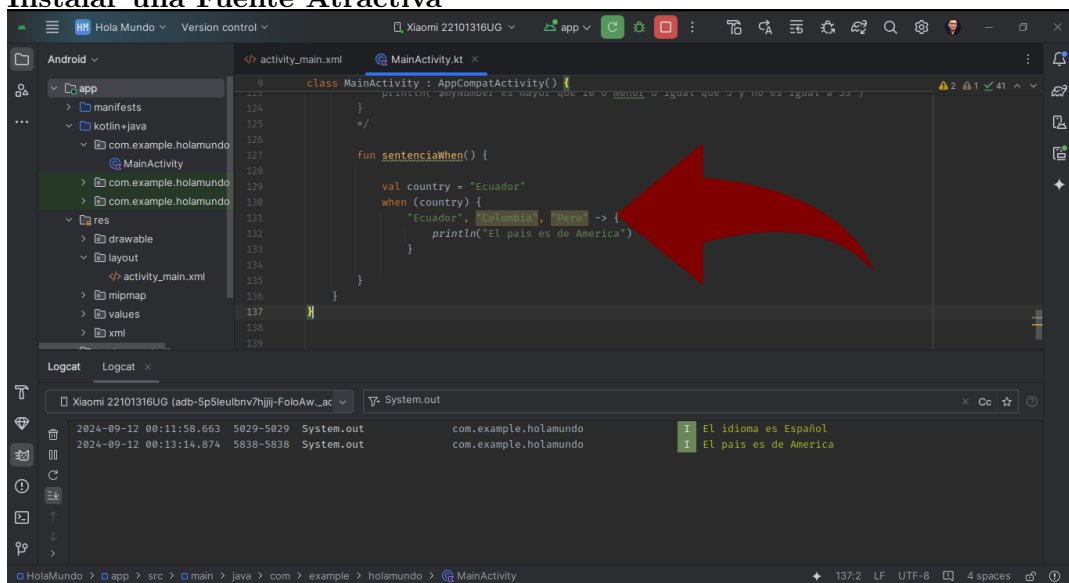
Esto nos permitirá ver solo la salida de nuestro programa y no la salida de la ejecución del código de la celda. De esta forma no se mostrarán los demás mensajes que se generan al ejecutar el código.

Por ejemplo:



Tip

Instalar una Fuente Atractiva



Una fuente muy atractiva para aprender a programar en Kotlin es FiraCode que es una fuente monoespaciada con ligaduras que facilitan la lectura del código. Puedes descargarla desde [aquí](#).

Para instalarla en tu editor de código favorito, sigue las instrucciones en la página de descarga. En Microsoft Windows puedes hacer doble clic en el archivo descargado y luego hacer clic en “Instalar”.

En mi caso como es Gnu/Linux Fedora 40 lo instalo con el siguiente comando:

```
sudo dnf install fira-code-fonts
```

Y luego selecciono la fuente en mi editor de código, para hacerlo en Android Studio

sigue los siguientes pasos:

1. Ve a **File > Settings > Editor > Font**.
2. Haz clic en el botón de selección de fuente y selecciona **Fira Code**.
3. Haz clic en **Apply** y luego en **OK**.

¡Listo! Ahora podrás disfrutar de una fuente de código más legible y atractiva.

¡Ahora ya sabes cómo declarar variables en Kotlin y utilizarlas en tus programas!

¡Sigue practicando y aprendiendo más sobre Kotlin para convertirte en un experto desarrollador de aplicaciones Android!

10 Tipos de Datos en Kotlin

En Kotlin, los tipos de datos se utilizan para definir el tipo de valores que pueden almacenar las variables. Los tipos de datos en Kotlin se dividen en dos categorías principales: tipos de datos primitivos y tipos de datos de objetos.

10.1 Tipos de Datos Primitivos

Los tipos de datos primitivos en Kotlin son tipos de datos básicos que representan valores simples como números enteros, números de punto flotante, caracteres y valores booleanos. Los tipos de datos primitivos en Kotlin son los siguientes:

- **Int**: Representa números enteros de 32 bits.

Ejemplo:

```
var edad: Int = 25
```

En este caso, la variable **edad** es de tipo **Int** (entero) y tiene un valor inicial de **25**.

- **Long**: Representa números enteros de 64 bits.

Ejemplo:

```
var poblacion: Long = 1000000
```

En este caso, la variable **poblacion** es de tipo **Long** (entero largo) y tiene un valor inicial de **1000000**.

- **Short**: Representa números enteros de 16 bits.

Ejemplo:

```
var cantidad: Short = 100
```

En este caso, la variable **cantidad** es de tipo **Short** (entero corto) y tiene un valor inicial de **100**.

- **Byte**: Representa números enteros de 8 bits.

Ejemplo:

```
var edad: Byte = 25
```

En este caso, la variable **edad** es de tipo **Byte** (entero de 8 bits) y tiene un valor inicial de **25**.

- **Float**: Representa números de punto flotante de 32 bits.

Ejemplo:

```
var precio: Float = 10.5f
```

En este caso, la variable **precio** es de tipo **Float** (número de punto flotante) y tiene un valor inicial de **10.5**.

- **Double**: Representa números de punto flotante de 64 bits.

Ejemplo:

```
var pi: Double = 3.1416
```

En este caso, la variable **pi** es de tipo **Double** (número de punto flotante doble) y tiene un valor inicial de **3.1416**.

- **Char**: Representa un carácter Unicode de 16 bits.

Ejemplo:

```
var letra: Char = 'A'
```

En este caso, la variable **letra** es de tipo **Char** (carácter) y tiene un valor inicial de **'A'**.

- **Boolean**: Representa un valor booleano verdadero o falso.

Ejemplo:

```
var activo: Boolean = true
```

En este caso, la variable **activo** es de tipo **Boolean** (booleano) y tiene un valor inicial de **true**.

Ahora vamos a trabajar con todos estos tipos de datos en un ejemplo práctico.

```

fun main() {
    // Variables mutables
    var edad: Int = 25
    var poblacion: Long = 1000000
    var cantidad: Short = 100
    var precio: Float = 10.5f
    var pi: Double = 3.1416
    var letra: Char = 'A'
    var activo: Boolean = true

    // Variables inmutables
    val gravedad: Double = 9.81

    // Mostrar los valores de las variables
    println("Edad: $edad")
    println("Población: $poblacion")
    println("Cantidad: $cantidad")
    println("Precio: $precio")
    println("Pi: $pi")
    println("Letra: $letra")
    println("Activo: $activo")
    println("Gravedad: $gravedad")
}

```

💡 Tip

Para poder ejecutar el bloque anterior es necesario llamar a la función **main()**. Esto se hace automáticamente en un proyecto de Kotlin, pero en este caso es necesario hacerlo manualmente.

The screenshot shows an IDE window with the following Kotlin code in MainActivity.kt:

```

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        fun main() {
            // Mostrar los valores de las variables
            println("Edad: $edad")
            println("Población: $poblacion")
            println("Cantidad: $cantidad")
            println("Precio: $precio")
            println("Pi: $pi")
            println("Letra: $letra")
            println("Activo: $activo")
            println("Gravedad: $gravedad")
        }
        main()
    }
}

```

The Logcat window at the bottom shows the output of the program:

```

2024-09-11 12:32:05.163 10142-10142 System.out com.example.holamundo I Edad: 25
2024-09-11 12:32:05.163 10142-10142 System.out com.example.holamundo I Población: 1000000
2024-09-11 12:32:05.164 10142-10142 System.out com.example.holamundo I Cantidad: 100
2024-09-11 12:32:05.164 10142-10142 System.out com.example.holamundo I Precio: 10.5
2024-09-11 12:32:05.164 10142-10142 System.out com.example.holamundo I Pi: 3.1416
2024-09-11 12:32:05.164 10142-10142 System.out com.example.holamundo I Letra: A
2024-09-11 12:32:05.164 10142-10142 System.out com.example.holamundo I Activo: true
2024-09-11 12:32:05.164 10142-10142 System.out com.example.holamundo I Gravedad: 9.81

```

Red arrows in the image highlight the `main()` call in the code and the corresponding log output.

En este ejemplo, se declaran variables mutables e inmutables de diferentes tipos de datos primitivos y se imprimen los valores

Tip

Como puedes observar el símbolo **\$** se utiliza para concatenar variables en una cadena de texto.

10.2 Tipos de Datos de Objetos

Los tipos de datos de objetos en Kotlin son tipos de datos que representan objetos complejos como cadenas de texto, arreglos, listas y mapas. Los tipos de datos de objetos en Kotlin son los siguientes:

- **String**: Representa una secuencia de caracteres.

Ejemplo:

```
var nombre: String = "Juan"
```

En este caso, la variable **nombre** es de tipo **String** (cadena de texto) y tiene un valor inicial de **“Juan”**.

- **Array**: Representa un arreglo de elementos del mismo tipo.

Ejemplo:

```
var numeros: Array<Int> = arrayOf(1, 2, 3, 4, 5)
```

En este caso, la variable **numeros** es de tipo **Array** (arreglo) de enteros y tiene un valor inicial de **[1, 2, 3, 4, 5]**.

- **List**: Representa una lista de elementos ordenados.

Ejemplo:

```
var colores: List<String> = listOf("Rojo", "Verde", "Azul")
```

En este caso, la variable **colores** es de tipo **List** (lista) de cadenas de texto y tiene un valor inicial de **[“Rojo”, “Verde”, “Azul”]**.

- **Map**: Representa un mapa de pares clave-valor.

Ejemplo:

```
var precios: Map<String, Double> = mapOf("Manzana" to 1.5, "Plátano" to 2.0, "Naranja" to 1.8)
```

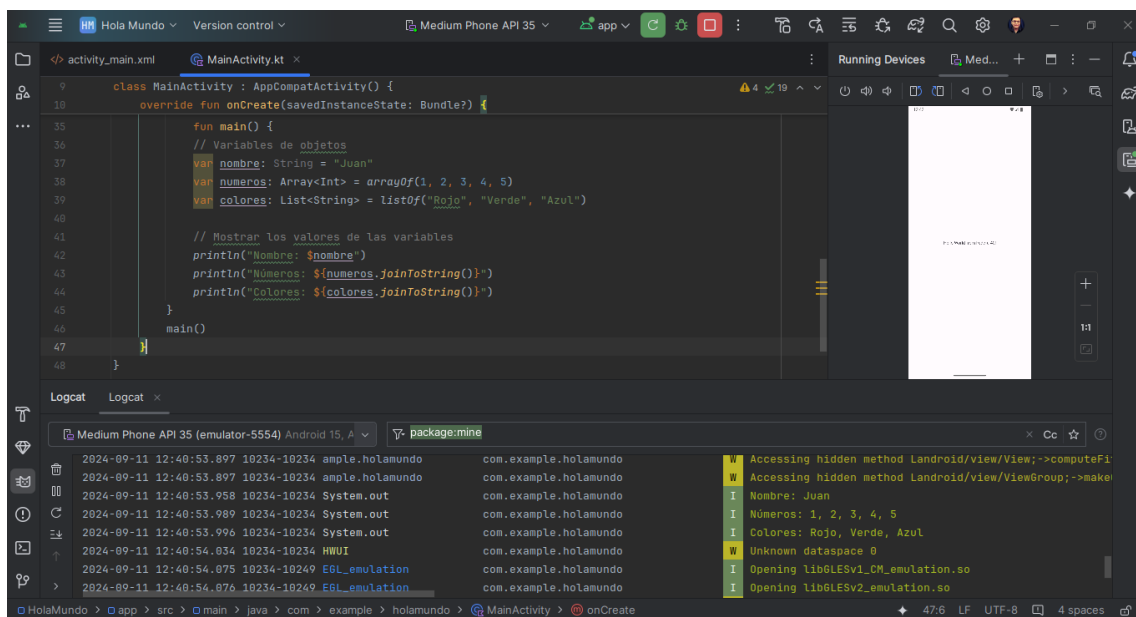
En este caso, la variable **precios** es de tipo **Map** (mapa) de cadenas de texto y números de punto flotante y tiene un valor inicial de {"Manzana"=1.5, "Plátano"=2.0, "Naranja"=1.0}.

Ahora vamos a trabajar con todos estos tipos de datos en un ejemplo práctico.

```
fun main() {
    // Variables de objetos
    var nombre: String = "Juan"
    var numeros: Array<Int> = arrayOf(1, 2, 3, 4, 5)
    var colores: List<String> = listOf("Rojo", "Verde", "Azul")

    // Mostrar los valores de las variables
    println("Nombre: $nombre")
    println("Números: ${numeros.joinToString()}")
    println("Colores: ${colores.joinToString()}")
}
```

En este ejemplo, se declaran variables de objetos de diferentes tipos de datos y se imprimen los valores.



10.3 Conversión de Tipos de Datos

En Kotlin, es posible convertir un tipo de dato a otro utilizando funciones de conversión. Las funciones de conversión en Kotlin son las siguientes:

- **toInt():** Convierte un valor a un entero.

Ejemplo:

```
var numero: Double = 10.5
var entero: Int = numero.toInt()
```

En este caso, la variable **entero** es de tipo **Int** y tiene un valor de **10**.

- **toDouble()**: Convierte un valor a un número de punto flotante.

Ejemplo:

```
var entero: Int = 10
var numero: Double = entero.toDouble()
```

En este caso, la variable **numero** es de tipo **Double** y tiene un valor de **10.0**.

- **toString()**: Convierte un valor a una cadena de texto.

Ejemplo:

```
var numero: Int = 10
var texto: String = numero.toString()
```

En este caso, la variable **texto** es de tipo **String** y tiene un valor de **"10"**.

- **toCharArray()**: Convierte una cadena de texto a un arreglo de caracteres.

Ejemplo:

```
var nombre: String = "Juan"
var caracteres: CharArray = nombre.toCharArray()
```

En este caso, la variable **caracteres** es de tipo **CharArray** y tiene un valor de **['J', 'u', 'a', 'n']**.

Ahora vamos a trabajar con todas estas funciones de conversión en un ejemplo práctico.

```
fun main() {
    // Conversión de tipos de datos
    var numero: Double = 10.5
    var entero: Int = numero.toInt()

    var entero2: Int = 10
    var numero2: Double = entero2.toDouble()

    var numero3: Int = 10
    var texto: String = numero3.toString()

    var nombre: String = "Juan"
    var caracteres: CharArray = nombre.toCharArray()
}
```

```

// Mostrar los valores de las variables
println("Número: $numero, Entero: $entero")
println("Entero2: $entero2, Número2: $numero2")
println("Número3: $numero3, Texto: $texto")
println("Nombre: $nombre, Caracteres: ${caracteres.joinToString()}")
}

```

En este ejemplo, se convierten valores de un tipo de dato a otro utilizando funciones de conversión y se imprimen los valores.

The screenshot shows an IDE with the following Kotlin code in MainActivity.kt:

```

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        fun main() {
            var numero3: Int = 10
            var texto: String = numero3.toString()

            var nombre: String = "Juan"
            var caracteres: CharArray = nombre.toCharArray()

            // Mostrar los valores de las variables
            println("Número: $numero, Entero: $entero")
            println("Entero2: $entero2, Número2: $numero2")
            println("Número3: $numero3, Texto: $texto")
            println("Nombre: $nombre, Caracteres: ${caracteres.joinToString()}")
        }

        main()
    }
}

```

The Logcat window shows the following output:

```

2024-09-11 12:46:01.117 10319-10319 System.out com.example.holamundo I Número: 10.5, Entero: 10
2024-09-11 12:46:01.118 10319-10319 System.out com.example.holamundo I Entero2: 10, Número2: 10.0
2024-09-11 12:46:01.118 10319-10319 System.out com.example.holamundo I Número3: 10, Texto: 10
2024-09-11 12:46:01.177 10319-10319 System.out com.example.holamundo I Nombre: Juan, Caracteres: J, u, a, n

```

¡Felicidades! Ahora sabes cómo trabajar con tipos de datos en Kotlin y cómo convertir valores de un tipo de dato a otro.

11 Convenciones de codificación en Kotlin

Para escribir código limpio y legible en Kotlin, es importante seguir ciertas convenciones de codificación que faciliten la comprensión del código y la colaboración con otros desarrolladores. A continuación, se presentan algunas convenciones de codificación recomendadas para Kotlin:

11.1 Nombres de variables

- Utiliza nombres descriptivos para las variables que reflejen su propósito y significado.

Ejemplo:

```
var nombreUsuario: String = "Juan"  
var edadUsuario: Int = 25  
var alturaUsuario: Double = 1.75
```

En este caso, los nombres de las variables reflejan claramente su propósito y significado.

- Utiliza el formato camelCase para los nombres de variables, comenzando con minúscula y utilizando mayúsculas para separar palabras.

Ejemplo:

```
var nombreUsuario: String = "Juan"  
var edadUsuario: Int = 25  
var alturaUsuario: Double = 1.75
```

11.2 Nombres de funciones

- Utiliza nombres descriptivos para las funciones que reflejen su propósito y acción.

Ejemplo:

```
fun saludarUsuario(nombre: String) {  
    println("¡Hola, $nombre!")  
}  
  
fun calcularEdad(anioNacimiento: Int): Int {  
    return 2022 - anioNacimiento  
}
```



```

}

fun convertirKilometrosAMetros(kilometros: Double): Double {
    return kilometros * 1000
}

```

En este caso, los nombres de las funciones reflejan claramente su propósito y acción.

- Utiliza el formato camelCase para los nombres de funciones, comenzando con minúscula y utilizando mayúsculas para separar palabras.

Ejemplo:

```

fun saludarUsuario(nombre: String) {
    println("¡Hola, $nombre!")
}

fun calcularEdad(anioNacimiento: Int): Int {
    return 2022 - anioNacimiento
}

fun convertirKilometrosAMetros(kilometros: Double): Double {
    return kilometros * 1000
}

```

11.3 Comentarios

- Utiliza comentarios para explicar el propósito y funcionamiento del código.

Ejemplo:

```

// Función que saluda al usuario
fun saludarUsuario(nombre: String) {
    println("¡Hola, $nombre!")
}

// Función que calcula la edad a partir del año de nacimiento
fun calcularEdad(anioNacimiento: Int): Int {
    return 2022 - anioNacimiento
}

// Función que convierte kilómetros a metros
fun convertirKilometrosAMetros(kilometros: Double): Double {
    return kilometros * 1000
}

```

En este caso, los comentarios explican el propósito y funcionamiento de las funciones.

- Utiliza comentarios en el código para documentar las partes importantes y facilitar su comprensión.

Ejemplo:

```
// Función que saluda al usuario
fun saludarUsuario(nombre: String) {
    println(";Hola, $nombre!")
}

// Función que calcula la edad a partir del año de nacimiento
fun calcularEdad(anioNacimiento: Int): Int {
    return 2022 - anioNacimiento
}

// Función que convierte kilómetros a metros
fun convertirKilometrosAMetros(kilometros: Double): Double {
    return kilometros * 1000
}
```

11.4 Indentación

- Utiliza una indentación consistente para mejorar la legibilidad del código.

Ejemplo:

```
fun main() {
    // Bloque de código
    if (condicion) {
        // Bloque de código
    } else {
        // Bloque de código
    }
}
```

En este caso, la indentación clara y consistente facilita la comprensión del código.

- Utiliza espacios en lugar de tabulaciones para la indentación del código.

Ejemplo:

```
fun main() {
    // Bloque de código
    if (condicion) {
        // Bloque de código
    } else {
        // Bloque de código
    }
}
```

```
}  
}
```

11.5 Conclusiones

Al seguir estas convenciones de codificación en Kotlin, podrás escribir código limpio, legible y fácil de mantener. Esto facilitará la colaboración con otros desarrolladores y mejorará la calidad y eficiencia de tu código.

¡Sigue practicando y mejorando tus habilidades de programación en Kotlin!

12 Comentarios en Kotlin

Los comentarios son una parte importante de cualquier programa, ya que permiten documentar el código y explicar su propósito y funcionamiento. En Kotlin, existen dos tipos de comentarios: de una sola línea y de varias líneas.

12.1 Comentarios de una sola línea

Los comentarios de una sola línea se utilizan para explicar una línea de código o parte de ella. Para escribir un comentario de una sola línea en Kotlin, se utiliza el símbolo `//` seguido del texto del comentario.

Ejemplo:

```
// Este es un comentario de una sola línea  
  
var edad: Int = 25 // Esta variable almacena la edad de una persona
```

En este caso, el comentario `// Esta variable almacena la edad de una persona` explica el propósito de la variable `edad`.

12.2 Comentarios de varias líneas

Los comentarios de varias líneas se utilizan para explicar bloques de código o secciones completas de un programa. Para escribir un comentario de varias líneas en Kotlin, se utiliza la secuencia `/*` para iniciar el comentario y `*/` para finalizarlo.

Ejemplo:

```
/*  
Este es un comentario de varias líneas  
que explica el propósito de este bloque de código  
*/  
  
fun saludarUsuario(nombre: String) {  
    println("¡Hola, $nombre!")  
}
```

En este caso, el comentario de varias líneas explica el propósito de la función `saludarUsuario`.

12.3 Comentarios de documentación

Los comentarios de documentación se utilizan para generar documentación automática a partir del código fuente. Para escribir un comentario de documentación en Kotlin, se utiliza la secuencia `/**` para iniciar el comentario y `*/` para finalizarlo.

Ejemplo:

```
/**
 * Esta función saluda al usuario
 * @param nombre El nombre del usuario
 */
fun saludarUsuario(nombre: String) {
    println("¡Hola, $nombre!")
}
```

En este caso, el comentario de documentación describe la función `saludarUsuario` y su parámetro `nombre`.

Los comentarios son una herramienta poderosa que te permite documentar tu código y facilitar su comprensión. Utiliza comentarios de una sola línea, comentarios de varias líneas y comentarios de documentación para explicar el propósito y funcionamiento de tu código en Kotlin.

¡Felicidades! Ahora sabes cómo utilizar comentarios en Kotlin para documentar tu código y facilitar su comprensión.

13 Sentencias if en Kotlin

En Kotlin, la sentencia **if** se utiliza para tomar decisiones basadas en una condición. La sintaxis básica de la sentencia **if** es la siguiente:

```
if (condicion) {  
    // Código a ejecutar si la condición es verdadera  
}
```

En este caso, si la condición es verdadera, se ejecuta el bloque de código dentro de las llaves. Si la condición es falsa, el bloque de código no se ejecuta.

13.1 Ejemplo de sentencia if

A continuación se muestra un ejemplo de cómo utilizar la sentencia **if** en Kotlin:

```
fun main() {  
    val edad = 18  
  
    if (edad >= 18) {  
        println("Eres mayor de edad")  
    }  
}
```

En este caso, si la variable **edad** es mayor o igual a **18**, se imprime el mensaje “Eres mayor de edad”.

13.2 Sentencia if-else

Además de la sentencia **if**, Kotlin también proporciona la sentencia **if-else** para tomar decisiones basadas en una condición. La sintaxis básica de la sentencia **if-else** es la siguiente:

```
if (condicion) {  
    // Código a ejecutar si la condición es verdadera  
} else {  
    // Código a ejecutar si la condición es falsa  
}
```

En este caso, si la condición es verdadera, se ejecuta el bloque de código dentro del primer conjunto de llaves. Si la condición es falsa, se ejecuta el bloque de código dentro del segundo conjunto de llaves.

13.3 Ejemplo de sentencia if-else

A continuación se muestra un ejemplo de cómo utilizar la sentencia **if-else** en Kotlin:

```
fun main() {
    val edad = 16

    if (edad >= 18) {
        println("Eres mayor de edad")
    } else {
        println("Eres menor de edad")
    }
}
```

En este caso, si la variable **edad** es mayor o igual a **18**, se imprime el mensaje “Eres mayor de edad”. De lo contrario, se imprime el mensaje “Eres menor de edad”.

13.4 Sentencia if-else-if

Además de la sentencia **if** y la sentencia **if-else**, Kotlin también proporciona la sentencia **if-else-if** para tomar decisiones basadas en múltiples condiciones. La sintaxis básica de la sentencia **if-else-if** es la siguiente:

```
if (condicion1) {
    // Código a ejecutar si la condición1 es verdadera
} else if (condicion2) {
    // Código a ejecutar si la condición2 es verdadera
} else {
    // Código a ejecutar si ninguna de las condiciones anteriores es verdadera
}
```

En este caso, se evalúan las condiciones en orden y se ejecuta el bloque de código correspondiente a la primera condición verdadera. Si ninguna de las condiciones es verdadera, se ejecuta el bloque de código dentro del conjunto de llaves **else**.

13.5 Ejemplo de sentencia if-else-if

A continuación se muestra un ejemplo de cómo utilizar la sentencia **if-else-if** en Kotlin:

```

fun main() {
    val edad = 16

    if (edad >= 18) {
        println("Eres mayor de edad")
    } else if (edad >= 13) {
        println("Eres adolescente")
    } else {
        println("Eres niño")
    }
}

```

En este caso, si la variable **edad** es mayor o igual a **18**, se imprime el mensaje “Eres mayor de edad”. Si la variable **edad** es mayor o igual a **13** pero menor que **18**, se imprime el mensaje “Eres adolescente”. De lo contrario, se imprime el mensaje “Eres niño”.

Ahora vamos a ver otros ejemplos para reforzar el uso de las sentencias **if**, **if-else** y **if-else-if** en Kotlin.

```

// Crear una función que determine si un número es positivo, negativo o cero

fun determinarSigno(numero: Int) {
    if (numero > 0) {
        println("El número es positivo")
    } else if (numero < 0) {
        println("El número es negativo")
    } else {
        println("El número es cero")
    }
}

fun main() {
    determinarSigno(5) // El número es positivo
    determinarSigno(-3) // El número es negativo
    determinarSigno(0) // El número es cero
}

```

En este caso, la función **determinarSigno** toma un número como argumento y determina si es positivo, negativo o cero utilizando una sentencia **if-else-if**.

```

// Crear una función que determine si un número es par o impar

fun determinarParidad(numero: Int) {
    if (numero % 2 == 0) {
        println("El número es par")
    } else {
        println("El número es impar")
    }
}

```



```

}

fun main() {
    determinarParidad(4) // El número es par
    determinarParidad(7) // El número es impar
}

```

En este caso, la función **determinarParidad** toma un número como argumento y determina si es par o impar utilizando una sentencia **if**.

```

// Crear una función que determine si un año es bisiesto o no

fun determinarBisiesto(anio: Int) {
    if (anio % 4 == 0) {
        if (anio % 100 != 0 || anio % 400 == 0) {
            println("El año es bisiesto")
        } else {
            println("El año no es bisiesto")
        }
    } else {
        println("El año no es bisiesto")
    }
}

fun main() {
    determinarBisiesto(2020) // El año es bisiesto
    determinarBisiesto(2021) // El año no es bisiesto
}

```

En este caso, la función **determinarBisiesto** toma un año como argumento y determina si es bisiesto o no utilizando una sentencia **if-else-if** anidada.

```

// Crear una función que determine el mayor de tres números

fun determinarMayor(num1: Int, num2: Int, num3: Int) {
    if (num1 >= num2 && num1 >= num3) {
        println("El mayor número es $num1")
    } else if (num2 >= num1 && num2 >= num3) {
        println("El mayor número es $num2")
    } else {
        println("El mayor número es $num3")
    }
}

fun main() {
    determinarMayor(5, 3, 7) // El mayor número es 7
    determinarMayor(10, 20, 15) // El mayor número es 20
}

```

En este caso, la función **determinarMayor** toma tres números como argumentos y determina cuál es el mayor utilizando una sentencia **if-else-if**.

Las sentencias **if**, **if-else** y **if-else-if** son fundamentales para tomar decisiones en un programa y controlar el flujo de ejecución del código en Kotlin. Es importante comprender cómo utilizar estas sentencias para escribir programas más complejos y funcionales.

¡Ahora puedes practicar la creación de tus propias funciones con sentencias **if**, **if-else** y **if-else-if** en Kotlin!

14 Sentencia when en Kotlin

En Kotlin, la sentencia **when** se utiliza para tomar decisiones basadas en el valor de una expresión. La sintaxis básica de la sentencia **when** es la siguiente:

```
when (expression) {
    valor1 -> {
        // Código a ejecutar si la expresión es igual a valor1
    }
    valor2 -> {
        // Código a ejecutar si la expresión es igual a valor2
    }
    valor3 -> {
        // Código a ejecutar si la expresión es igual a valor3
    }
    else -> {
        // Código a ejecutar si la expresión no coincide con ningún valor
    }
}
```

En este caso, la sentencia **when** evalúa la expresión y ejecuta el bloque de código correspondiente al valor que coincide con la expresión. Si la expresión no coincide con ningún valor, se ejecuta el bloque de código dentro de la cláusula **else**.

14.1 Ejemplo de sentencia when

A continuación se muestra un ejemplo de cómo utilizar la sentencia **when** en Kotlin:

```
// Función que imprime el día de la semana

fun imprimirDiaSemana(dia: Int) {
    when (dia) {
        1 -> println("Lunes")
        2 -> println("Martes")
        3 -> println("Miércoles")
        4 -> println("Jueves")
        5 -> println("Viernes")
        6 -> println("Sábado")
        7 -> println("Domingo")
        else -> println("Día inválido")
    }
}
```

```

}

fun main() {
    imprimirDiaSemana(1)
    imprimirDiaSemana(4)
    imprimirDiaSemana(7)
    imprimirDiaSemana(9)
}

```

En este caso, la función **imprimirDiaSemana** toma un parámetro **dia** y utiliza la sentencia **when** para imprimir el día de la semana correspondiente al valor del parámetro. Si el valor del parámetro no coincide con ningún día de la semana, se imprime “Día inválido”.

La sentencia **when** es una forma concisa y legible de tomar decisiones basadas en el valor de una expresión en Kotlin. Utiliza la sentencia **when** para simplificar la lógica de tus programas y mejorar su mantenibilidad.

Ahora vamos a programar otros ejemplos para practicar el uso de la sentencia **when** en Kotlin.

```

// Crear una función que determine el tipo de día (laboral o festivo)

fun determinarTipoDia(dia: String) {
    when (dia) {
        "Lunes", "Martes", "Miércoles", "Jueves", "Viernes" -> println("Día laboral")
        "Sábado", "Domingo" -> println("Día festivo")
        else -> println("Día inválido")
    }
}

fun main() {
    determinarTipoDia("Lunes") // Día laboral
    determinarTipoDia("Sábado") // Día festivo
    determinarTipoDia("Miércoles") // Día laboral
    determinarTipoDia("Viernes") // Día laboral
    determinarTipoDia("Domingo") // Día festivo
    determinarTipoDia("Martes") // Día laboral
    determinarTipoDia("Jueves") // Día laboral
    determinarTipoDia("Vacaciones") // Día inválido
}

```

En este caso, la función **determinarTipoDia** toma un parámetro **dia** y utiliza la sentencia **when** para determinar si es un día laboral, festivo o inválido en función del valor del parámetro.

La sentencia **when** es una herramienta poderosa que te permite tomar decisiones basadas en el valor de una expresión de forma concisa y legible en Kotlin. Utiliza la sentencia **when** para simplificar la lógica de tus programas y mejorar su mantenibilidad.

```
// Crear una función que determine el tipo de triángulo (equilátero, isósceles o escaleno)

fun determinarTipoTriangulo(lado1: Int, lado2: Int, lado3: Int) {
    when {
        lado1 == lado2 && lado2 == lado3 -> println("Triángulo equilátero")
        lado1 == lado2 || lado2 == lado3 || lado1 == lado3 -> println("Triángulo isósceles")
        else -> println("Triángulo escaleno")
    }
}

fun main() {
    determinarTipoTriangulo(5, 5, 5) // Triángulo equilátero
    determinarTipoTriangulo(3, 4, 3) // Triángulo isósceles
    determinarTipoTriangulo(7, 4, 5) // Triángulo escaleno
}
```

En este caso, la función **determinarTipoTriangulo** toma tres parámetros **lado1**, **lado2** y **lado3** que representan los lados de un triángulo y utiliza la sentencia **when** para determinar si es equilátero, isósceles o escaleno en función de las longitudes de los lados.

La sentencia **when** es una forma concisa y legible de tomar decisiones basadas en múltiples condiciones en Kotlin. Utiliza la sentencia **when** para simplificar la lógica de tus programas y mejorar su mantenibilidad.

```
// Crear una función que determine el horario de una tienda

fun determinarHorario(tiempo: Int) {
    when (tiempo) {
        in 0..6 -> println("La tienda está cerrada")
        in 7..12 -> println("La tienda está abierta por la mañana")
        in 13..18 -> println("La tienda está abierta por la tarde")
        in 19..23 -> println("La tienda está abierta por la noche")
        else -> println("Hora inválida")
    }
}

fun main() {
    determinarHorario(5) // La tienda está cerrada
    determinarHorario(10) // La tienda está abierta por la mañana
    determinarHorario(15) // La tienda está abierta por la tarde
    determinarHorario(20) // La tienda está abierta por la noche
    determinarHorario(25) // Hora inválida
}
```

En este caso, la función **determinarHorario** toma un parámetro **tiempo** que representa la hora del día y utiliza la sentencia **when** para determinar el horario de la tienda en función de la hora del día.

La sentencia **when** es una herramienta poderosa que te permite tomar decisiones basadas en rangos de valores en Kotlin. Utiliza la sentencia **when** para simplificar la lógica de tus programas y mejorar su mantenibilidad.

```
// Crear una función que determine el signo zodiacal

fun determinarSignoZodiacal(dia: Int, mes: Int) {
    when {
        (mes == 3 && dia >= 21) || (mes == 4 && dia <= 19) -> println("Aries")
        (mes == 4 && dia >= 20) || (mes == 5 && dia <= 20) -> println("Tauro")
        (mes == 5 && dia >= 21) || (mes == 6 && dia <= 20) -> println("Géminis")
        (mes == 6 && dia >= 21) || (mes == 7 && dia <= 22) -> println("Cáncer")
        (mes == 7 && dia >= 23) || (mes == 8 && dia <= 22) -> println("Leo")
        (mes == 8 && dia >= 23) || (mes == 9 && dia <= 22) -> println("Virgo")
        (mes == 9 && dia >= 23) || (mes == 10 && dia <= 22) -> println("Libra")
        (mes == 10 && dia >= 23) || (mes == 11 && dia <= 21) -> println("Escorpio")
        (mes == 11 && dia >= 22) || (mes == 12 && dia <= 21) -> println("Sagitario")
        (mes == 12 && dia >= 22) || (mes == 1 && dia <= 19) -> println("Capricornio")
        (mes == 1 && dia >= 20) || (mes == 2 && dia <= 18) -> println("Acuario")
        (mes == 2 && dia >= 19) || (mes == 3 && dia <= 20) -> println("Piscis")
        else -> println("Fecha inválida")
    }
}

fun main() {
    determinarSignoZodiacal(21, 3) // Aries
    determinarSignoZodiacal(20, 4) // Tauro
    determinarSignoZodiacal(21, 6) // Géminis
    determinarSignoZodiacal(23, 7) // Leo
    determinarSignoZodiacal(22, 8) // Virgo
    determinarSignoZodiacal(22, 12) // Capricornio
    determinarSignoZodiacal(19, 1) // Capricornio
    determinarSignoZodiacal(20, 1) // Acuario
    determinarSignoZodiacal(18, 2) // Acuario
    determinarSignoZodiacal(19, 2) // Piscis
    determinarSignoZodiacal(20, 3) // Piscis
    determinarSignoZodiacal(32, 13) // Fecha inválida
}
```

En este caso, la función **determinarSignoZodiacal** toma dos parámetros **dia** y **mes** que representan la fecha de nacimiento de una persona y utiliza la sentencia **when** para determinar su signo zodiacal en función de la fecha de nacimiento.

La sentencia **when** es una forma concisa y legible de tomar decisiones basadas en múltiples condiciones en Kotlin. Utiliza la sentencia **when** para simplificar la lógica de tus programas y mejorar su mantenibilidad.

¡Excelente! Ahora sabes cómo utilizar la sentencia **when** en Kotlin para tomar decisiones basadas en el valor de una expresión. Utiliza la sentencia **when** para simplificar la lógica de tus programas y mejorar su mantenibilidad.