CFGS DESARROLLO DE APLICACIONES MULTIPLATAFORMA

UD 1. Introducción

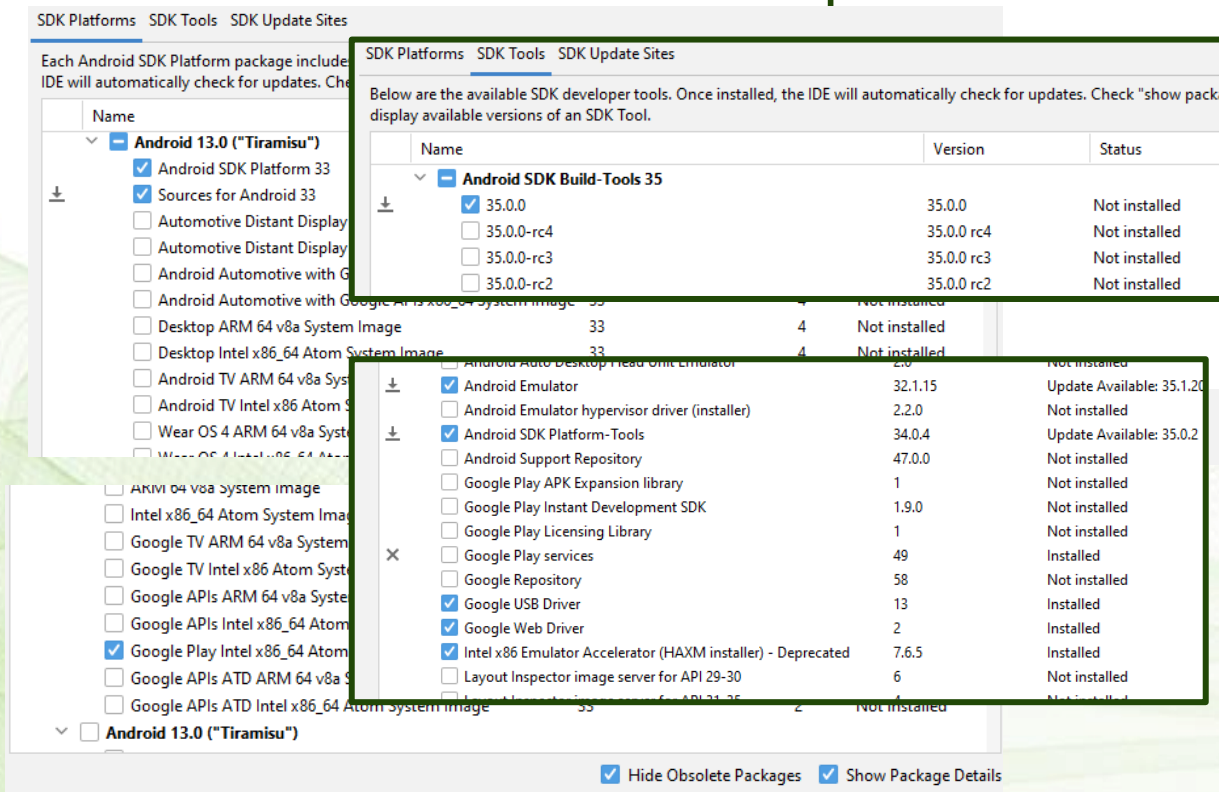
Módulo: Programación multimedia y dispositivos móviles

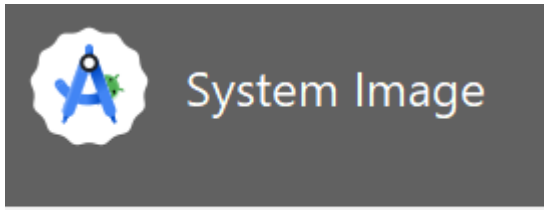
Víctor J. Vergel Rodríguez

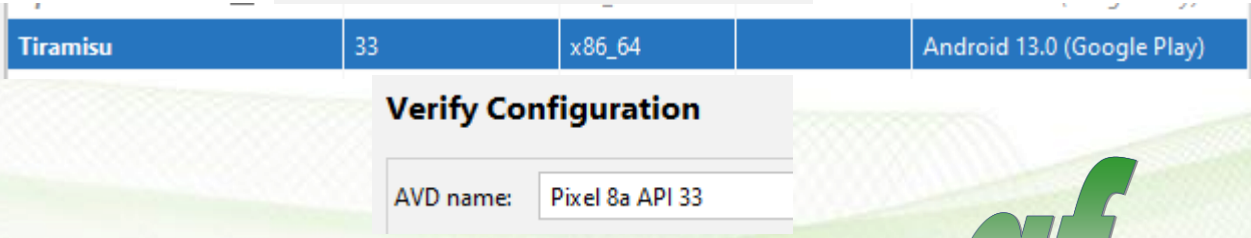
Centro de Enseñanza
Gregorio Fernández

Instalación

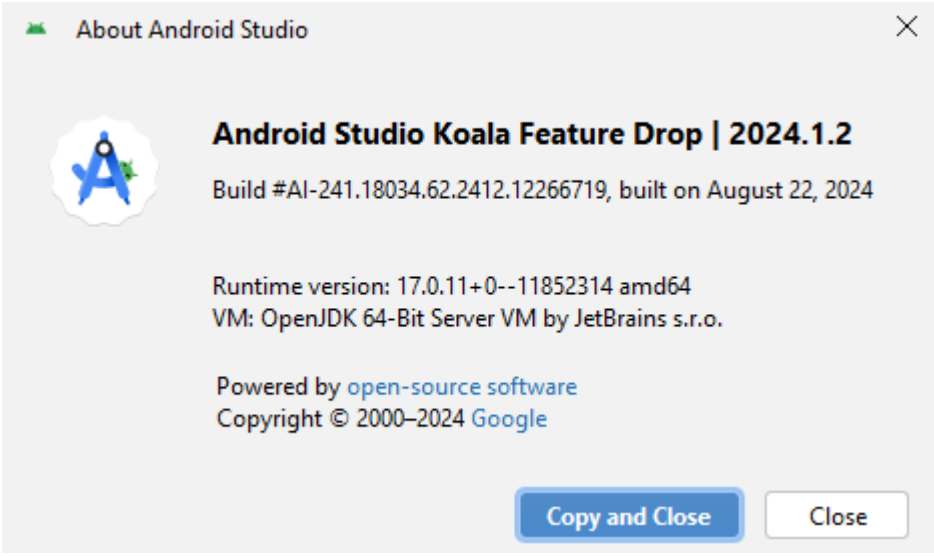
- Android Studio Koala | 2024.1.2







Introducción Android



- **Descarga:** https://developer.android.com/studio
- **API levels:** https://apilevels.com/
- **Documentación del API:** https://developer.android.com/reference/

GitHub

- `Add` -> Añadir ficheros al repositorio
- `Commit` – Confirmar cambios en el repositorio
- `New Branch` -> Creamos una nueva rama
- `Checkout` -> Cambiar a la nueva rama

GitHub

- https://github.com/victorvergelgregorio/Moviles2425.git

Introducción Android

- **Android:** Sistema Operativo basado en una versión modificada de un kernel de Linux. Existen algunas variantes como Android TV, Android Auto, usadas en diferentes tipos de dispositivos. Open Source
- **Android SDK (Software Development Kit):** Conjunto de herramientas para desarrollar aplicaciones para la arquitectura Android. Android Studio (IDE), SDK Manager, Android Virtual Device Manager (AVD), diferentes versiones y sus APIS, NDK (para escribir en C o C++)…
- **Android SDK Platforms Tools**: Conjunto de herramientas adicionales al Android SDK como, por ejemplo, **adb** (Android Debug Bridge), utilizada para ejecutar comandos en un dispositivo Android conectado al equipo, **fastboot** para cargar imágenes de móviles, **sqlite3** trabajando con base de datos.

Introducción Android

- **Jetpack**. Es una colección de bibliotecas, herramientas y guías de buenas prácticas desarrolladas por Google para simplificar y acelerar el proceso de desarrollo de aplicaciones Android. Incluye: ViewModel, LiveData, Navigation,Data binding…
- **apk:** Android Package, básicamente el equivalente a un exe de los instaladores de las aplicaciones windows
- **Art: (Android Runtime)** Entorno de ejecución de Android, remplaza a Dalvik, máquina virtual de Android que permite la ejecución de aplicaciones.
- **Google Play.** Publicamos nuestras apps con cuenta de desarrollador. Nuestras apps tendrán un identificador único asociado al nombre del paquete

Introducción Android

- **Activity:** Concepto similar a lo que es una ventana/pantalla de la aplicación.
- **Fragment:** Cada vez más en auge. Interfaces más reutilizables
- **Layouts y views.** Componentes básicos de un interfaz, serían los contenedores y los componentes de esos contenedores
- **Intent.** Objeto que comparte información entre Activitys.
- **Application.** Es una clase base que representa la instancia global de la aplicación y se crea cuando se inicia la aplicación y se destruye cuando la aplicación se cierra por completo

Kotlin

- Aparece en 2016 con JetBrains (Desarrolladores de IntelliJ IDEA). En el evento Google I/O del año 2019, Google empezó a apostar por Kotlin.

- **Ventajas:**
  - **Expresivo y conciso**: Hacer más con menos. En Kotlin se reduce el código.
  - **Código más seguro –Safe-**: Salva del NullpointerException
  - **Corrutinas**: Permite trabajar de forma asincrónica de una forma mucho más sencilla.

- Para poder practicar Kotlin nos encontramos con:

- Playground: https://play.kotlinlang.org
- Kotlin ejemplos: https://play.kotlinlang.org/byExample/overview"
- Tutorial: https://kotlinlang.org/docs/home.html

Introducción Android Studio

- **Teclado Android Studio**

Pulsando **mayúscula mayúscula** te sale una ventana en la cual puedes buscar archivos y te los abre rápidamente.

Alt+F7 muestra donde se utilizan una variable que previamente ha seleccionado

Ctrl+D duplicamos una línea.

Alt+ flecha arriba o abajo me muevo rápido entre funciones.

Ctrl – sobre una función, contraigo la función

Alt+ derecha/izquierda me muevo por las pestañas

Ctrl+mayusc+F12 maximiza el espacio de la ventana de código o restaura el resto de ventanas

Ctrl+Alt +e visualiza tus últimas localizaciones del cursos

Ctrl+Mayusc+ supr si quieres quitar código envolvente

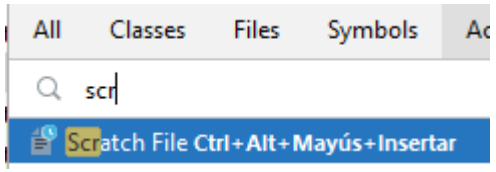
Alt+J en los XML voy cambiando múltiples selecciones.

Ctrl+Q sobre una función me muestra la ayuda sobre los métodos de llamada de la misma

Ctrl+/ o Ctrl+Mayus+/ (la del teclado numérico) comentar varias líneas

Ejecución en Android Studio

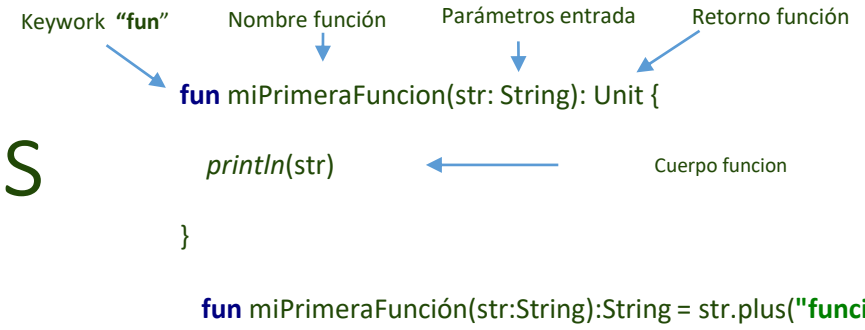
- CTRL+ALT+MAYs+N -> Actions -> Scratch o directamente Ctrl+Alt+Mayúsc+Insertar



Kotlin - variables

- Las variables inmutables o finales se definen con “val”
- Las variables mutables o que pueden cambiar se definen con “var”:
- Las variables pueden tiparse explícitamente o ser inferido

Kotlin - funciones



- Funciones **normales**
- Funciones de **orden superior** , reciben de parámetro una función

```
println("operacion: ${operacion(5,3,::suma)}")
println("operacion: ${operacion(5,3,::resta)}")

fun suma(a:Int,b:Int):Int {
    return a+b
}
fun resta(a:Int,b:Int):Int {
    return a-b
}
fun operacion(a:Int,b:Int,operacion_fn:(Int,Int)->Int):Int {
    return operacion_fn(a,b)
}
```

Kotlin – funciones especiales

- En Kotlin, podemos meter por parámetro a una función otra:

```
fun ejecutarOperacion(operacion: (Int, Int) -> Int, a: Int, b: Int): Int {
    return operacion(a, b)
}
```

- Funciones lambda:

```
val variable = { argumentos -> cuerpo }
```

Kotlin – variables especiales

- val variable: TipoDato by lazy { ... } - será inicializada una variable por la expresión proporcionada en el bloque { ... } la primera vez que se acceda a ella, y luego conservará ese valor en las siguientes llamadas.
- Observable. La función observable toma dos parámetros:
  - El valor inicial de la propiedad.
  - Una lambda que se ejecutará cada vez que la propiedad cambie:
    - ✓ **prop**: El objeto que está siendo observado (en este caso, nombre)
    - ✓ **old**: El valor anterior de la propiedad.
    - ✓ **new**: El nuevo valor de la propiedad.

```
var nombre: String by Delegates.observable("Sin nombre") { prop, old, new ->
    println("$old -> $new")
}
```

- Lateinit. propiedades que no necesitan ser inicializadas de inmediato en el constructor o en la declaración, pero que se inicializarán antes de su primer uso.

Kotlin – Null safe

- Kotlin es un lenguaje null-safe, por ello, todas las variables deben estar inicializadas.
- Las variables nulas se definen mediante “Opcionales”

```
var nameUser:String = null => no compila
var nameUser:String? = null  => es un opcional
var nameUser?.isNotEmpty() => debe tener el opcional para acceder de forma segura a su método.
```

- Desempaquetar de forma segura un opcional. En Kotlin para comprobar la propiedad de valor deberemos utilizar el operador “let” para desempaquetar de forma segura la propiedad sin que produzca el NullPointer. Nunca utilizar “!!” para obtener la propiedad de la variable, sin comprobar la nulidad previamente.

```
var nameUser:String? = null  => es un opcional
nameUser?.let{ valor_property ->  la propiedad no es nula    }
```

Kotlin – Tipos Set y MutableSet

- Tanto Set como MutableSet son interfaces que representan colecciones que no permiten elementos duplicados. Set no admiten operaciones de modificación como agregar, eliminar o actualizar

```
val mutableSet: MutableSet<Int> = mutableSetOf(1, 2, 3)
mutableSet.add(4)
mutableSet.remove(2)
println(mutableSet) // Imprime: [1, 3, 4]
```

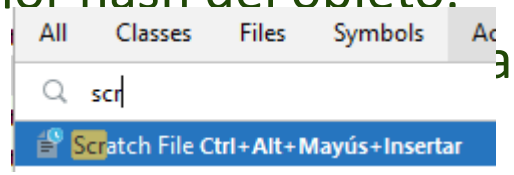
- En el caso de val mutableSet: la variable es inmutable, lo que significa que su referencia no puede cambiar a otro objeto de tipo MutableSet<Int>, pero los elementos dentro del conjunto sí pueden ser modificados ya que es un conjunto mutable.
- Métodos: sort, groupBy, find

Kotlin – Clase Any

- Se trata de la jerarquía de tipos equivalente a Object de java. Si no se especifica explícitamente un tipo para una variable en Kotlin, se asume Any por defecto.
- Singleton. Un objecto que **solo tiene una única instancia** en toda la aplicación y proporciona un punto de acceso global para acceder a sus miembros.

Kotlin – Dataclass

- Automáticamente genera métodos implícitos
  - **`equals()`:** Para comparar objetos por igualdad estructural.
  - **`hashCode()`:** Para calcular el valor hash del objeto.
  - **`toString()`:** Para devolver una cadena de texto del objeto.
  - **`copy()`:** Copiar objetos



- **Constructor primario implícito**: acepta todos los parámetros definidos en la lista de propiedades de la clase.
- **Propiedades inmutables o mutables**: Puedes definir las propiedades de una data class como inmutables (val) o mutables (var).
- **No se heredan métodos y propiedades de la superclase Any**: Las data class no heredan los métodos ni las propiedades de la clase base Any

Kotlin – Programación orientada a objetos

- Por defecto no se puede heredar de una clase, son finales, hay que abrirla.
- // Clase base

```
dataclass Person(val name: String, val age: Int)
```

- // Clase que hereda de una data classdata

```
class Employee(val position: String, val salary: Double, val name: String, val age: Int)
        fun main() {
        val employee = Employee("Developer", 50000.0, "John", 30)
        println(employee)
        // Output: Employee(position=Developer, salary=50000.0, name=John, age=30)
}
```

- En Kotlin, las data classes pueden heredar de otras clases y también pueden ser clases base para otras clases. Sin embargo, hay algunas restricciones:
  - Las data classes pueden heredar de otras data classes.
  - Si una clase hereda de una data class, debe ser una data class también.
  - Las data classes no pueden heredar de clases no data.

Kotlin – Visibilidad

Modifier	Description
public	visible everywhere
private	visible inside the same class only
internal	visible inside the same module
protected	visible inside the same class and its subclasses

Kotlin – Clases internas

```
Data class ClaseExterna {
    // Código de la clase externa

    inner class ClaseInterna {
        // Código de la clase interna
    }
}
```