Introducción a Kotlin

Índice

keep coding

Pero antes...

Probando Kotlin Online

No es necesario instalar nada probar el lenguaje Kotlin
Prueba Kotlin Online

El IDE: Intellij

Descarga e Instalación

Descarga

Descárgalo en:
https://www.jetbrains.com/es-es/idea/

¡Atención con las versiones! utilizaremos la versión **Community**

Anticipando problemas

Intellij incluye un SDK de Java
No requiere una instalación separada del SDK, ni de las típicas variables del entorno que conlleva

¿Qué es un SDK?

El **SDK** de Java (Java Software Development Kit) es un conjunto de herramientas que permite desarrollar, compilar y ejecutar aplicaciones en Java.

- Incluye:
  - El JRE (Java Runtime Environment)
  - El compilador javac
  - Las bibliotecas estándar
  - Otras utilidades necesarias para programar en Java

keep coding

Java

¿Java?

¡¡Pero esto no va de Kotlin!!

Let’s work together!

Configuración básica

Documentación

Encuentra la documentación oficial aquí

Creando nuestro primer proyecto

Encuentra la documentación oficial aquí

Gradle

Gradle es una herramienta de **automatización de compilación** (build tool) utilizada principalmente para proyectos de software.

Permite:

- Compilar código
- Gestionar dependencias
- Ejecutar pruebas
- Generar archivos ejecutables
- Otras tareas (Ejemplo, flavours).

Gradle

Pros:

- Versatilidad
- Facilidad para gestionar proyectos

Contras:

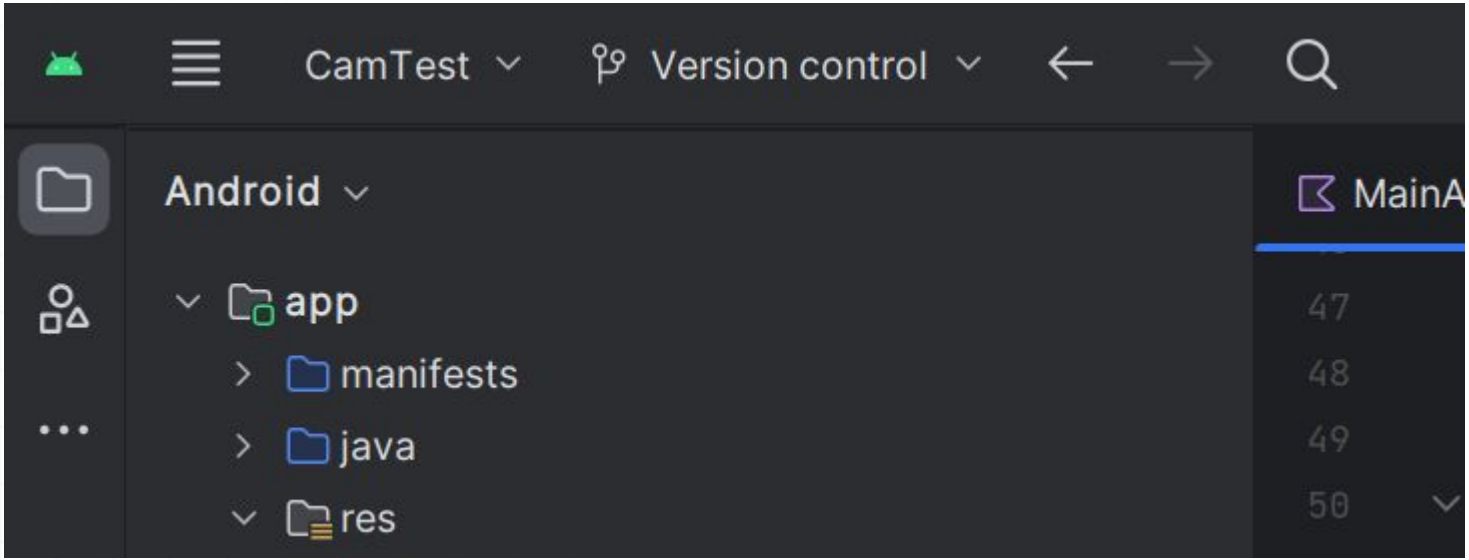
- Aprendizaje
- Errores poco claros y fatales
- Lentitud

Ejecutando nuestro primer proyecto

Encuentra la documentación oficial aquí

Paso Opcional

Poniéndolo… ¿Más útil?



keep coding

Programando en Kotlin

Variables

Metáfora

Las variables son cómo cajas… ¡se queda algo corta!

En kotlin se entiende mejor como la “libreta del gestor del almacén” + “las cajas”

Tipos Básicos

- Booleanos
- Números: Enteros, decimales…
- Letras
- Cadenas de Texto
- Arrays

Objetos

Las clases son una combinación de **variables** y **funciones** que denominaremos **atributos** y **métodos**

Todo son objetos

La diferencia entre los tipos básicos y los objetos realmente radica en la **inmutabilidad** de los elementos de dentro del objeto

Variables Inmutables

Operaciones Básicas

- Crear una variable y asignarle valor
- Realizar acciones sobre las variables
- Comprar tipos y/o asumir tipos
- Cambiar de un valor de un tipo a otro
- Gestionar los valores nulos

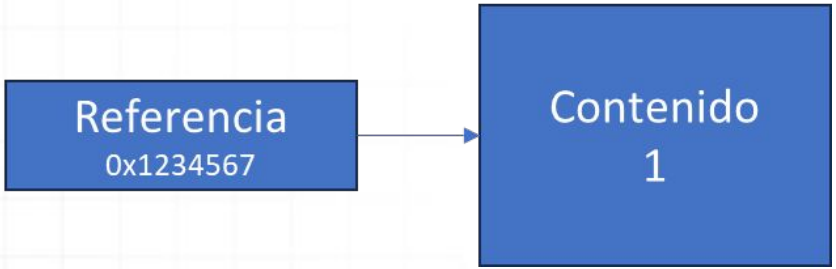
Trabajando

keep coding

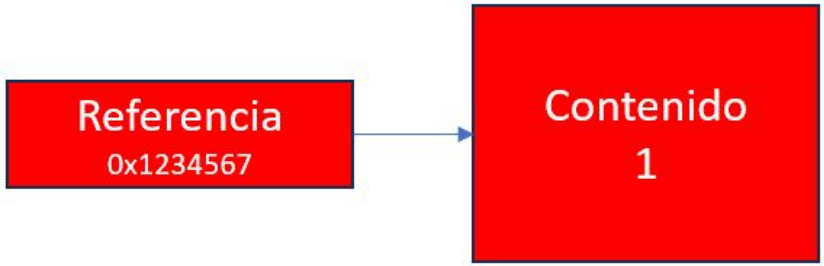
- Concatenar un String con unas variables

Variables inmutables

Var num = 1



Val num = 1



Documentación

Sintaxis de Variables
Tipos básicos

Ejemplo

https://github.com/KeepCodingMobile19/kotlin/commit/f7ef83f08348a02ad25e8ccfc3b8624a245c56f3

https://github.com/KeepCodingMobile19/kotlin/commit/6ce14d4905e26d2c36ba4bc9cc3f5474c1ac330d

Paseando por Intellij

Acciones

Organización de los ficheros en un proyecto de Kotlin con Gradle
Navegando entre ficheros
Crear nuevos ficheros “main”
Eligiendo que “main” ejecutar

Conceptos

Iniciar el Debugger
Agregar Breakpoints
Observar la secuencia de ejecución

Primeros Pasos con el Debugger

Observando el valor de las variables

¡Y cambiando los valores en vivo!

Funciones

Operaciones

Crear una función…

- simple
- que devuelve datos
- con parámetros
- con parámetros opcionales

Ejemplo

https://github.com/KeepCodingMobile19/kotlin/commit/e10f3372ec106040b669efadc0a1ef63f1a6d9ab

https://github.com/KeepCodingMobile19/kotlin/commit/22c0337d592eb88b3937e109166e1d12e568086c

Tipos Básicos

Ejemplos:
Ejemplo básico funciones (ej1)
Ejemplo más completo (ej2)

Ejercicio

Trabajando con funciones:

- Añade una nueva función Main
- Crea una función que reciba dos String, los transforme a Integer, los sume y devuelva un Double. Llámala “sumaStrings”.
- Escribe el resultado por pantalla. Asume que los String se transforman en Int sin problemas.

Solución

https://github.com/KeepCodingMobile19/kotlin/commit/821c99e703c03767bd977567f95a6298af8ee449

Ejercicio 1B

- Crea una función que reciba dos Strings, nombre y apellido.
- La función debe escribir por pantalla: Hola, soy “nombre” y me apellido es “apellido” (donde “nombre” y “apellido” son los valores recibidos en las función)
- Crea una función que reciba dos Int, num1 y num2 y devuelva un String. La función debe sumarlos y devolver un string con el resultado.

Solución

Tipos

Ejercicio:
Propuesta ejercicio (ej3)
Solucion (ej3Sol)

Estructuras condicionales

keep coding

Pero antes...

Comparadores 1

Igualdad: “==”

- variable1 == variable2

Mayor: “>”. Mayor o igual: “>=”. Menor: “<”. Menor o igual: “<=”

- variable1 > variable2

Pertenencia a un tipo: is

- variable is Long

Pertenencia a un rango: in

- variable in 1..10

Comparadores 2

AND: “&&”
- Ambas condiciones deben cumplirse
- variable1 == 1 && variable2 == 10

OR: “||”
- Al menos una de las condiciones deben cumplirse
- variable1 == 1 || variable2 == 10

NEGATION: “!”
- Niega el operador siguiente
- variable !is Long - La variable no debe ser un Long
- variable != 1 - La variable debe ser distinta a !=1

Estructuras

keep coding

if ()

If () else

when ()

when

Ejemplo

keep coding

Ejercicios

Ejercicio

Crea un proyecto nuevo de Kotlin en IntelliJ.

- Crea una función que determine qué precio deben pagar para el abono.

Anuales / Una sola zona

Zona/ Abono	A	B1	B2	B3	C1	C2	E1	E2
Normal	546,00€	637,00€	720,00€	820,00€	820,0€	820,00€	1.106,00€	1.318,00€
Joven	200,00€							
Tercera Edad	Gratuito						-	

☞ En todos los tipos de Abono Transporte existen **modalidades con descuento** dirigidas a los siguientes colectivos:

- Familias Numerosas (categoría General y Especial)
- Personas con Discapacidad ≥ 65%

Ejercicio

Crea un proyecto nuevo de Kotlin en IntelliJ.
Crea una función que reciba un Integer y un String, siendo el Integer la vida de un personaje y el String su nombre. Llámala “iniciarPersonaje”. La función debe devolver un String que contenga:

- Si la vida es menos 20 puntos, entonces debe devolver: “Precaución {Nombre}, estás bajo vida”
- Si la persona tiene entre 21 y 80, entonces debe devolver: “{Nombre}, estás algo herido”
- Si la persona tiene más de 81, entonces debe devolver: “{Nombre}, estás en perfectas condiciones”

Ejercicio

Realiza el ejercicio anterior utilizando un when.

Solución

Bucles

Bucles

repeat(){}
while()
do - while(condición)
for ()

Ejemplo

keep coding

Ejercicios

keep coding

Pero antes…

Leer una variable por teclado

Para el ejercicio necesitaremos que el usuario escriba un dato por consola.
Para ello tenemos la función readLine() que nos devuelve un String?, posteriormente habrá que transformarlo al tipo deseado.

Ejercicio

Complementando al ejercicio 2 y utilizando los bucles vistos hasta ahora, realiza un programa en Kotlin que solamente permita establecer una vida comprendida entre 0 y 100. Si el número no cumple con estas condiciones, deberá volver a preguntarlo.

Solución

Arrays y listas

Arrays

Los Arrays son colecciones de tamaño fijo de elementos.
Ejemplo, array de Strings:

Pos	0	1	2	3	4	5	6
Contenido	“A”	“B”	“C”	“D”	“E”	“F”	“G”

Creando arrays

- arrayOf()

Recorriendo

- forEach(){}
- forEachIndexed(){}

Ejemplo

Listas

Los Listas son colecciones de tamaño variable de elementos.

Pos	0	1	2	3	4	5	6
Contenido	“A”	“B”	“C”	“D”	“E”	“F”	“G”

Añadimos un elemento a la lista:

Pos	0	1	2	3	4	5	6	7
Contenido	“A”	“B”	“C”	“D”	“E”	“F”	“G”	“H”

Listas

Las listas son colecciones de tamaño variable de elementos.

val - List	var - List
val - MutableList	var - MutableList

Creando

- listOf()
- mutableListOf()

Recorriendo

- forEach(){}
- forEachIndexed(){}

Ejemplo

keep coding

Ejercicios

Ejercicio 4A

Crea una lista que:

- Parte 1:
  - Crea una lista “listaRandom” de 100 elementos compuesta de números aleatorios comprendidos entre el 0 y el 9.
  - Imprime la lista por pantalla
- Parte 2:
  - Crea una lista vacía “listaResultado” en cuya posición…
  - Posición 0 se debe contar cuantos 0 hay en la listaRandom.
  - Posición 1 se debe contar cuantos 1 hay en la listaRandom.
  - etc. con todos los números.

Ejercicio

Crea una lista de varios elementos
Utilizando únicamente lo visto hasta ahora y este código:

- val lista = listOf("H", 1, 2, "o", 0.2, "l", 1.0, "a", 0.3, "!")

Realiza un programa que:

- Sume todos los números de la lista en una misma variable (Integers y Doubles) e imprímelo.
- Concatena todos los String en un único String e imprímelo

Solución

Lambdas

Lambdas

Una Lambda es una forma resumida de declarar una función, simplemente utilizando los {}
Las Lambdas pueden:

- Asignarse a variables
- Ser / pasarse como parámetros en una función
- Ejecutarse cuando se desee

keep coding

Pero… ¡Esto me suena!

Lambdas usadas

- Repeat(X){ }
- lista.forEach{ }
- if { }

Ejemplo

keep coding

Ejercicios

Ejercicio 5A

Transforma las siguientes funciones en Lambdas y ejecútalas.

- fun lambda1() { println("Esta Lambda escribe: Hola") }
- fun lambda2(integer : Int) { println("Esta Lambda recibe un Int y lo escribe: $integer") }
- fun ejercicio3(posicion : Int, list : List<Any>) { println("Esta Lambda recibe un Int y una Lista. Escribe al elemento que haya en la posición ${ list[posicion]}") }

Solución

Clases

Clases

Las clases son una combinación de **variables** y **funciones** que denominaremos **atributos** y **métodos**

Los objetos son las variables creadas de una Clase

Atributos

Características:

- Privadas
- Públicas
- Otros modificadores

Métodos

Características:

- Privados
- Públicos
- Otros modificadores

Imprimiendo

¿Qué aparece al imprimir un objeto?

Solución:

- el método toString()

Pasando

Si queremos inicializar una clase, podemos pasarle parámetros como si de un método se tratase.

Ejemplo

https://github.com/KeepCodingMobile18/Kotlin/commit/55a2fef6df364562ebf62887581f4c4c7a60198b

Variables

Val num = MyInt(1)

Ejercicio

Crea una clase Personaje que reciba el nombre y la vida actual del personaje:

- Poder lanzarse “recibirGolpe()”, quitándole un número aleatorio de vida (entre 10 y 60 puntos). La vida no puede caer por debajo de 0.
- Poder lanzarse “recibirCuracion()”, añadiéndole 20 puntos de vida. La vida no puede superar los 100 puntos.
- Almacena en un listado todos los golpes y curaciones (curaciones en positivo como golpes en negativo).

Solución

Extras: Let

keep coding

Pero antes…

Problema de los nulos

Hasta ahora hemos visto la programación secuencial:

- En la programación concurrente se pueden encontrar nuevas casuísticas… que presentan nuevos desafíos.

Let

Let es una función muy usada en kotlin. Sus funcionamiento se

- Aplica sobre cualquier variable.
- Recibe como argumento una lambda.
  - La lambda solo se ejecuta si la variable no es null.
  - La Lambda recibe una copia de la variable no nullable.

Ejemplo

Otras funciones de Scope

Existen otras funciones similares:

- Run
- With
- Also
- Run
- Apply

keep coding

Resumen funciones de Scope

	it	this	return	extension fun
let	object	class	anything	yes
apply		object	object	yes
run		object	anything	yes
also	object	class	object	yes
with		object	anything	

Ejemplo

Data Class

Data

Una “Data Class” es una “Class” a la que se le añade/modifican ciertas funciones útiles:

- Equals() y Hash().
- toString().
- componenN().
- copy().

Requisitos

Para ser “Data Class”, es necesario cumplir cierto requisitos:

- El constructor primario necesita tener al menos un parámetro.
- Los parámetros del constructor deben estar marcados como var o val.
- Data class no puede ser abstract, open, sealed, or inner.

Ejercicio

Responde a las siguientes preguntas:
Preguntas 1
Preguntas 2

Ejemplo

Sealed Class

Data Class

keep coding

Una “Sealed Class” es una “Class” que es del tipo de una de las clases que le componen.

```
sealed class HeroResponse() {
class Hero(name: String,maxLive : Int = 100): HeroResponse()
class Error(val message: String): HeroResponse()
}
```

Ejemplo

Herencia e Interfaces

keep coding

Pero antes…

Abstract

Modificadores:

- open
- abtract
- override

Llamada al Padre. Super()

Ejemplo

Personaje:

1. Tiene nombre
2. Dice su nombre
3. Tiene un nivel de vida
4. Dice su nivel de vida

Las demás clases tienen lo mismo que la clase Personaje más lo que tengan propio

- Saiyajin:
  - Tiene cola
  - Tiene evolución
- Namekiano:
  - Tiene antenas
  - Pone huevos
- Humano:
  - Dice el número de veces que ha muerto
  - Dice el número de veces que ha resucitado

Ejemplo sin herencia

Crea una lista con todos las clases anteriores.

- Haz que todos digan su nombre recorriendo la lista
- Cuenta cuantos Namekianos, Sayiajins y Humanos (ejemplo 3 namek, 1 Humano).
- Ordena los elementos de las lista alfabéticamente por nombre

Ejercicio sin herencia

Crea una lista con todos las clases anteriores.

- Haz que todos digan su nivel de vida
- Cuenta cuántos tienen más de 50.
- Ordena los elementos de las lista por aquellos que tienen más vida

Ejemplos

Interface

Interface

Concepto: Crea un contrato (una serie de funciones o variables) que todas las clases que implementen esta interfaz deberán completar

Restricciones:

- No se pueden crear objeto de una interfaz

Alternativas

- Puedes crear objetos de la clase que herede de una interfaz
- Puedes crear un object de una interfaz

Ejemplo con interfaces

Crea una lista con todos las clases anteriores.

- Haz que todos digan hola recorriendo la lista
- Cuenta cuantos Namekianos, Sayiajins y Humanos (ejemplo 3 namek, 1 Humano).
- Ordena los elementos de las lista alfabéticamente por nombre

Ejercicio con interfaces

Crea una lista con todos las clases anteriores.

- Haz que todos digan su nivel de vida
- Cuenta cuántos tienen más de 50.
- Ordena los elementos de las lista por aquellos que tienen más vida

Ejercicio Extra

Partiendo del ejercicio anterior, modifícalo con lo siguiente:

- Los Héroes pueden ser de varios clases. Humanos los cuales pueden atacar y curar, los namekianos, los cuales pueden curarse, atacar y poner huevos y los Saiyans, los cuales pueden transformar en super guerreros.
- Después de poner un huevo, los namekianos tienen reducido el ataque un 50% durante 30 segundos.
- Un Saiyajin puede evolucionar, lo cual lo duplica la vida máxima y hace que sus ataques haga el doble del daño normal.

Ejemplos

Herencia

Herencia

Concepto: Se crea una clase con aquello que es común a todas las demás clases a la que denominaremos clase “Padre”. Las clases hijas tendrán todas las características de la clase “Padre” y podrán añadir / sustituir las suyas propias

Restricciones:

- No se puede heredar de más de una Padre

Alternativas

- Puedes crear interfaces para realizar algo parecido

Ejemplo con Herencia

Crea una lista con todos las clases anteriores.

- Haz que todos digan hola recorriendo la lista
- Cuenta cuantos Namekianos, Sayiajins y Humanos (ejemplo 3 namek, 1 Humano).
- Ordena los elementos de las lista alfabéticamente por nombre

Ejercicio con Herencia

Crea una lista con todos las clases anteriores.

- Haz que todos digan su nivel de vida
- Cuenta cuántos tienen más de 50.
- Ordena los elementos de las lista por aquellos que tienen más vida

Ejercicio Extra

Partiendo del ejercicio anterior, modifícalo con lo siguiente:

- Los Héroes pueden ser de varios clases. Humanos los cuales pueden atacar y curar, los namekianos, los cuales pueden curarse, atacar y poner huevos y los Saiyans, los cuales pueden transformar en super guerreros.
- Después de poner un huevo, los namekianos tienen reducido el ataque un 50% durante 30 segundos.
- Un Saiyajin puede evolucionar, lo cual lo duplica la vida máxima y hace que sus ataques haga el doble del daño normal.

Ejemplos

Abstract Class

Abstract

Concepto: Una clase que tiene sentido y lógica por su cuenta (hace cosas) de forma autónoma a la vez que le falta algún comportamiento por definir.

Restricciones:

- No se pueden crear objeto de la clase abstracta
- Solo puedes heredar de una clase abstracta a mismo tiempo

Alternativas

- Puedes crear objetos de la clase que herede de una clase abstracta
- Puedes crear un object de una clase abstracta

Ejemplo

Si la clase Personaje tuviera sentido por sí misma (hay personajes sueltos, sin ser otra cosa) sería un candidata a clase abstracta.

Ejercicio Extra

Partiendo del ejercicio anterior, modifícalo con lo siguiente:

- Los Heroes pueden ser de varios clases. Humanos los cuales pueden atacar y curar, los namekianos, los cuales pueden curarse, atacar y poner huevos y los Saiyans, los cuales pueden transformar en super guerreros.
- Después de poner un huevo, los namekianos tienen reducido el ataque un 50% durante 30 segundos.
- Un Saiyan puede evolucionar, lo cual lo duplica la vida máxima y hace que sus ataques haga el doble del daño normal.

Interface con cuerpo

Interface

Concepto: Un punto medio entre las clases abstractas y las interfaces. Tiene sentido para implementar comportamientos concretos complejos

Restricciones:

- No se pueden crear objeto de una interfaz con cuerpo

Alternativas

- Puedes crear objetos de la clase que herede de una interfaz con cuerpo
- Puedes crear un object de una interfaz con cuerpo

Ejemplo

Si la clase Personaje **no** tuviera sentido por sí misma (**no** hay personajes sueltos, sin ser otra cosa) sería un candidata a interfaz con cuerpo.

Ejercicio

Partiendo del ejercicio anterior, modifícalo con lo siguiente:

- Los Héroes pueden ser de varios clases. Humanos los cuales pueden atacar y curar, los namekianos, los cuales pueden curarse, atacar y poner huevos y los Saiyans, los cuales pueden transformar en super guerreros.
- Después de poner un huevo, los namekianos tienen reducido el ataque un 50% durante 30 segundos.
- Un Saiyan puede evolucionar, lo cual lo duplica la vida máxima y hace que sus ataques haga el doble del daño normal.

Extensión

¿Qué es?

Añade funcionalidad a una clase ya existente sin modificarla.

Ejemplo

Json

Json

Json en un formato de texto cuyo objetivo es facilitar el intercambio de datos, principalmente en la WEB.
Es útil para transformar clases a texto
Un Json luce:
{"nombre":"Juan", "edad":20}

Uso

Facilitar la transformación mediante una librería.
Gradle:
implementation("com.google.code.gson:gson:2.9.0")

Uso

De objeto a JSON:

```
val gson = Gson()
gson.toJson(Persona("Pedro", 32))
```

Resultado:

Uso

```
fun guardarJugador(){
    val archivo = File("Jugadores/$nombre.txt")
    archivo.writeText("personaje.toJson()")
}
```

```
fun cargarPersonaje(){
    try {
        val archivo = File("Jugadores/$nombre.txt")
        val json = archivo.readLines()
    } catch (exception : Exception) {
        println("Error la leer los datos.")
    }
}
```

Ejercicio 8

Modifica la clase Personaje realizada en el ejercicio 6 de tal manera que:
Añada una función a esa clase que permita guardar en un archivo de texto el JSON que corresponde a ese objeto.
Añada una función que, obteniendo la dirección de un fichero de texto, si contiene el JSON correspondiente a un Dado, cree un objeto con los detalles correspondientes.

Ejemplo

Test Básicos

Ejemplo

Corrutinas

Introducción

Manejar tareas asíncronas y concurrentes sin bloquear el hilo principal.
Son más eficientes que los hilos tradicionales - Menor consumo de recursos

¿Dónde se lanzan? Scope

¿Dónde se ejecuta la corrutina?

- GlobalScope
  - CorrutineScope
  - ViewModelScope, LifeCycleScope…

¿Cuanto dura?

- Lo que tarde en ejecutarse lo que hay en su interior

¿Cuando se cierra?

- Cuando termina de ejecutarse o cuando alguien la cancela
  - job.cancel()

¿Cómo se lanzan?

GlobalScope.Launch() - No devuelve nada
GlobalScope.Async() - Devuelve un deferred

Permiten un Scope!

Dispatchers

Dispatchers.Main → Hilo principal (UI).
Dispatchers.IO → Operaciones de entrada/salida (archivos, red).
Dispatchers.Default → Cálculos intensivos.

Suspend Function

keep coding

Función que requiere ser ejecutada en una corrutina

Ejemplo

Ejercicio

Necesitas simular dos procesos en paralelo:

- Descargar datos de un servidor (tarda 3 segundos).
- Procesar una imagen (tarda 2 segundos).

Usa corrutinas para que ambas tareas se ejecuten al mismo tiempo.

- Imprimir "Inicio del proceso" al comienzo.
- Mostrar mensajes cuando cada tarea inicia y cuando termina.
- Imprimir "Todas las tareas han finalizado" al final.

Warnings!

Corrutina no es un Hilo
Atención al uso de elementos de concurrencia de hilos en corrutinas!

Resumen

Son eficientes porque no bloquean hilos innecesariamente.
Se usan con launch y async, y pueden cambiar de contexto (Dispatchers).
Se pueden cancelar con Job.cancel().

Ejercicios extras

Clases y Herencia

¿Vemos un ejemplo? (ej6)

Ejercicio (ej7)

Herencia (ej8)

Colecciones y Arrays

List (ej25)

Filter (28)

Map (29)

Sorted (38)

Let

- Muy útil para null checking
- Se ejecuta sobre objetos, proporcionandonos un bloque
- Devuelve el resultado de la última expresión
- Accesible vía ‘it’ o con un nombre definido

¿Vemos un ejemplo? (ej17)

Funciones de Scope

Run

- Muy parecido a let
- La diferencia principal es que dentro del bloque accedemos a la referencia utilizando this (en lugar de it)
- Devuelve el resultado de la última expresión
- Útil cuando queremos llamar a métodos del objeto en lugar de pasarlos como argumento

¿Vemos un ejemplo? (ej18)

Aclaremos algunas cosas ... (ej19)

Funciones de Scope

With

- No es una extensión (es decir no se aplica sobre un objeto)
- Puede acceder a los miembros del argumento que recibe
- Podemos omitir el nombre de la instancia para acceder a sus miembros

¿Vemos un ejemplo? (ej20)

Funciones de Scope

keep coding

Apply

- Muy parecido a run
- Dentro del bloque accedemos a la referencia utilizando this
- Útil para inicializar objetos

¿Vemos un ejemplo? (ej21)

Volvamos al ejemplo de antes con el run y adaptemos a apply (ej22)

Funciones de Scope

keep coding

Also

- Muy parecido a apply
- Dentro del bloque accedemos a la referencia utilizando it
- Útil para añadir acciones adicionales

Vamos al ejemplo (ej23)

Funciones de Scope

Ejercicio Scope Functions (ej24)

Solución (ej24Sol)

Ejemplo de un mal uso de las funciones de scope

Fin

keep coding

Datos de contacto

LinkeIn de Carlos de Tena Bellmont