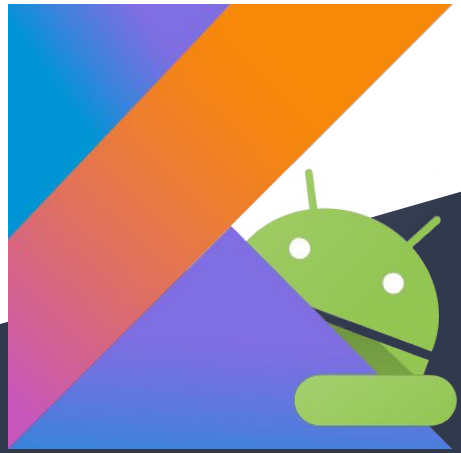
# Hilos – Executor – Corrutinas

Reflexión: 11-11-2021



**Integrantes:**
Sebastián Farías
Leonardo Rodenas

# Reflexión:

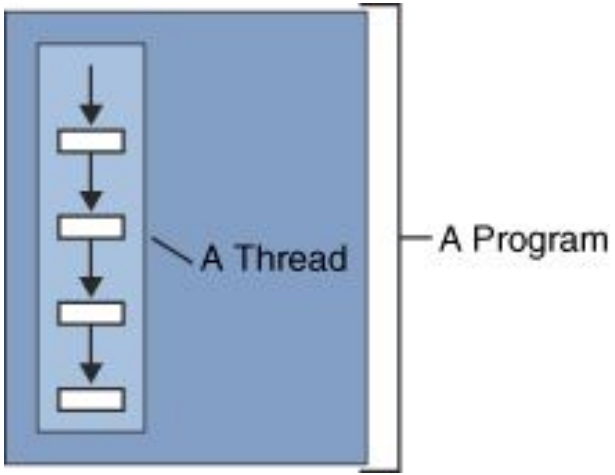
(Aclaración, dado que esto fue trabajo grupal Sebastián Farías comparte junto conmigo este ppt y evidencia)

**Reflexión:** El día de hoy fue una clase bastante buena, en lo personal me sirvió para consolidar conceptos que a priori parecen similares pero que se usan y funcionan diferentes.
Trabajamos en grupos y expusimos los conceptos obtenidos y solucionamos dudas y respondimos consultas a otros compañeros.

# ¿Qué es un Hilo?

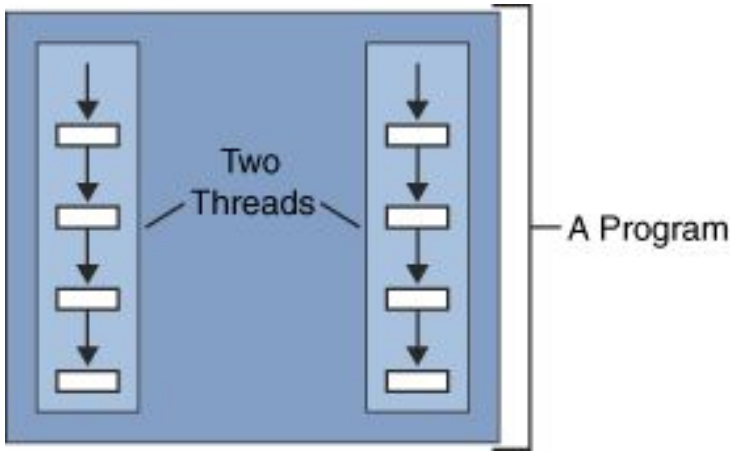
**Definición:** Un hilo es un único flujo secuencial de control dentro de un programa.

En cualquier momento durante el tiempo de ejecución del hilo, hay un único punto de ejecución. Sin embargo, un hilo en sí mismo no es un programa; un hilo no puede ejecutarse por sí solo. Más bien, se ejecuta dentro de un programa. La siguiente figura muestra esta relación.



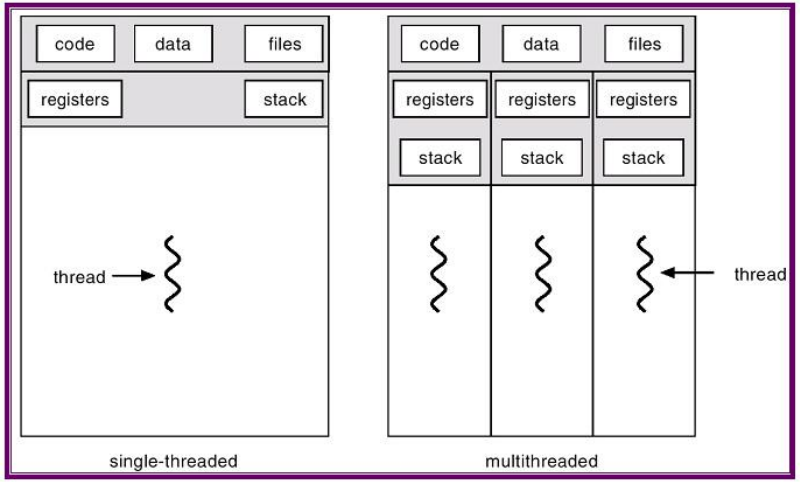
# ¿Qué es un Hilo?

**¿Su gracia?** → No manejar uno solo, manejar varios hilos o subprocesos que se ejecutan al mismo tiempo y realizan diferentes tareas en un solo programa.




**Ejemplo:** navegador o una App = aplicación multiproceso

# ¿Qué es un Hilo?

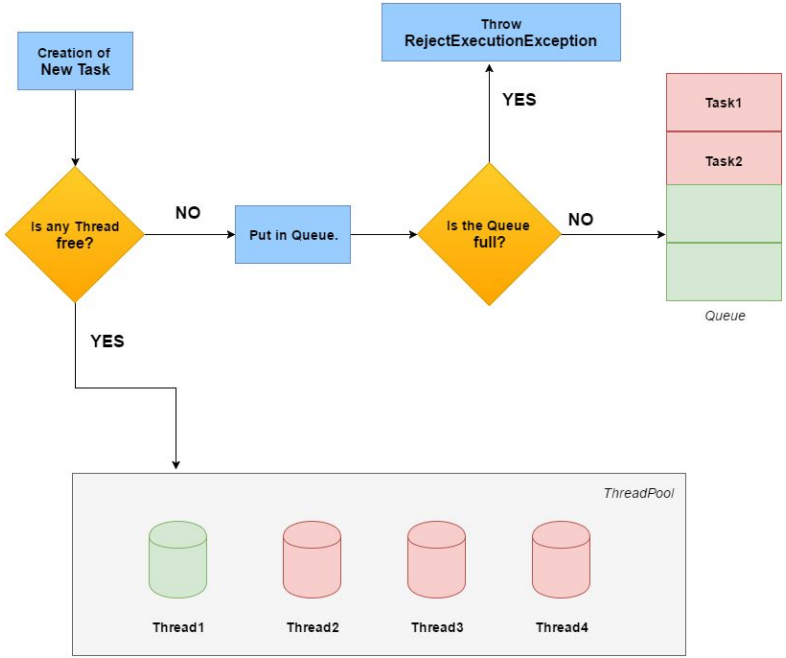



**¿Proceso ligero?** → Un hilo es similar a un proceso real en que ambos tienen un único flujo secuencial de control. Sin embargo, un hilo se considera ligero porque se ejecuta dentro del contexto de un programa completo y aprovecha los recursos asignados para ese programa y el entorno del programa.

Como flujo secuencial de control, un hilo debe crear algunos de sus propios recursos dentro de un programa en ejecución. Por ejemplo, un hilo debe tener su propia pila de ejecución (stack) y contador de programa (register).

# Executor

**¿Qué es?** → Objeto que ejecuta tareas ejecutables enviadas. Esta interfaz proporciona una forma de desacoplar el envío de tareas de la lógica de cómo se ejecutará cada tarea. Normalmente se utiliza un Executor en lugar de crear hilos explícitamente.




```
public static final ExecutorService wreiteInDB = Executors.newFixedThreadPool(4)
```

# Hilos

# vs

# Executor

```
new Thread(new RunnableTask()).start()
```

```
//Java
Executor executor = anExecutor();
  executor.execute(new RunnableTask1());


//kotlin

executor : Executor = anExecutor()
  executor.execute(RunnableTask1());
```

Arriba: crear un hilo

Abajo: Ejecuta el runnable sin asignar un hilo

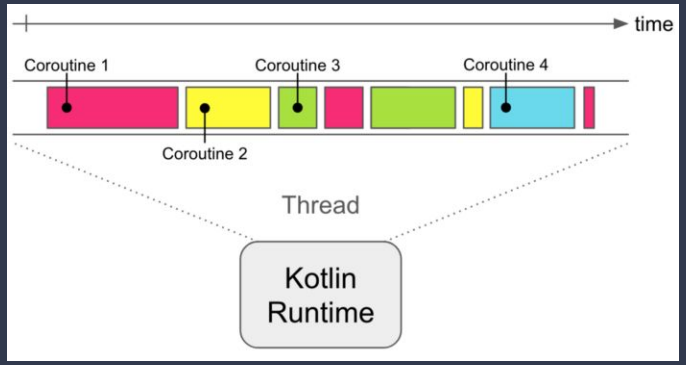
# Corrutina

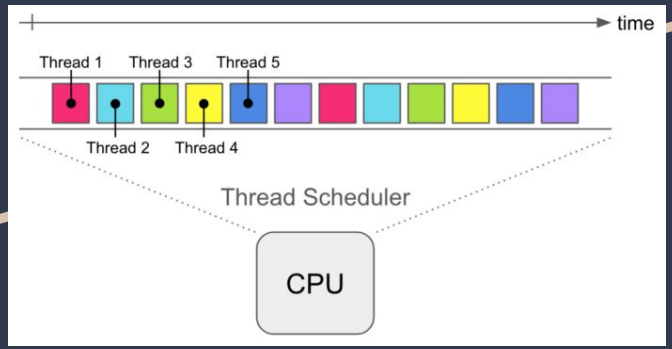
**¿Qué son las coroutines?**

Según la propia documentación oficial, una coroutine es en sí misma un hilo liviano. Es decir, se vale de hilos o pool de hilos para ejecutar código de manera concurrente (destino llegar a UI) y de forma muy eficiente optimizando el uso de recursos.

La creación de muchos hilos atenta contra el rendimiento en algunas situaciones, ya que estos subprocesos pueden imponer una carga excesiva sobre el “garbage collector” y el “Object Allocation” de la JVM.

# ¿Ventajas?




**Desarrollador** → Implementación de código asincrónico con un estilo idéntico al código sincrónico.

**Rendimiento** → permiten la ejecución de miles y hasta millones de subprocesos concurrentemente con un uso de recursos eficiente.

**Implementación** → manipulación sencilla de los hilos de ejecución valiéndose de los Dispatchers.

**Versatilidad** → las corrutinas se pueden suspender y reanudar a mitad de la ejecución.

# Elementos de las Corrutinas:

## Context

**Dispatcher:** Sistema que planifica la gestión de los trabajos y el inicio y fin de los procesos, estableciendo prioridades y asignando procesadores a cada tarea.

El contexto determina en qué hilo se ejecutarán las corrutinas. Hay cuatro opciones:

**Dispatchers.Default:** para trabajo intenso de CPU (por ejemplo, ordenar una lista grande)

**Dispatchers.Main:** Se limita al subproceso principal que opera con objetos de UI. Por lo general, estos despachadores son de un solo subproceso.Para trabajar con esto, se debe agregar la dependencia al proyecto:

```
org.jetbrains.kotlinx:kotlinx-coroutines-android:1.3.9
```

**Dispatcher.Unconfinned:** ejecuta corrutinas sin limitarlasa ningún hilo específico

**Dispatchers.IO:** para trabajos pesados de IO (por ejemplo, consultas de bases de datos de larga duración).

# Constructores

## launch

```
val job = GlobalScope.launch() {
    println("${Thread.currentThread()} has run." )
}
```

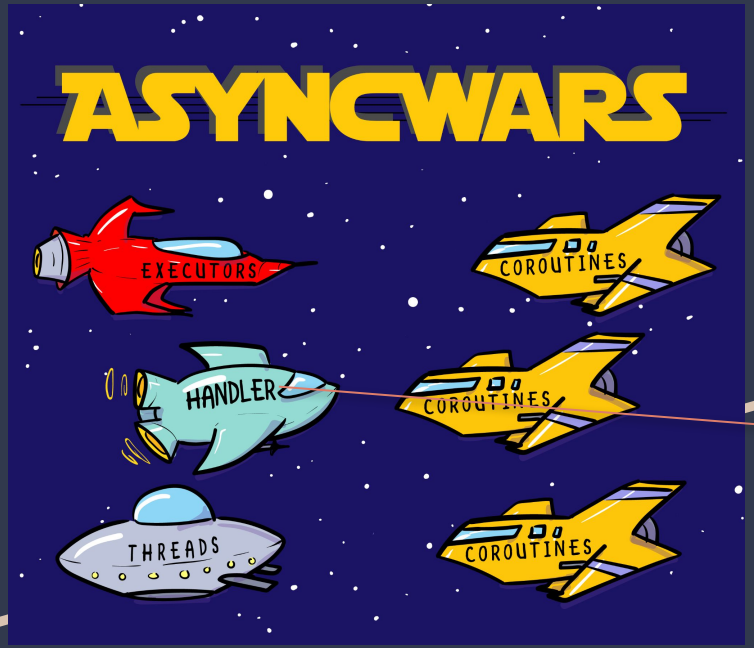
## async

```
val deferred = async (Dispatchers.Unconfined, CoroutineStart.LAZY) {
    return@async "${Thread.currentThread()} has run."
}
```

## runBlocking

```
runBlocking(newSingleThreadContext( "dedicatedThread")) {
    val result = deferred.await()
    println(result)
}
```

# Fuentes:




- https://medium.com/kotlin-en-android/coroutines-con-kotlin-constructores-de-coroutines-8a9e10c8187e
- https://medium.com/android-news/executor-framework-understanding-the-basics-43d575e72310
- https://developer.android.com/reference/kotlin/java/util/concurrent/Executor
- https://stackoverflow.com/questions/54416840/kotlin-coroutines-scope-vs-coroutine-context
- https://www.baeldung.com/kotlin/threads-coroutines
- https://github.com/Leonardo-Rodenas/Repaso/tree/main/Octubre/Clase15-10-2021%20Room