

{desafío}
latam_

Threads y Coroutines _

Parte I



Conceptos claves que estudiaremos

- Conocer qué es síncrono y asíncrono en la programación de software.
- Conocer qué son los threads.
- Conocer qué es el thread UI en Android y cómo aplicarlo.
- Conocer qué es AsyncTask y cómo aplicarlo.
- Conocer qué es un Service y cómo exponerlo en un proyecto android.
- Conocer qué es un AlarmManager y como crear una tarea programada.
- Comprender un proceso en Background y un proceso en MainThread/UI.
- Comprender qué es el patrón de promesas y como utilizarlo en Kotlin.
- Comprender el concepto de concurrencia.

Síncrono y Asíncrono

Tareas, procesos síncronos

Un proceso síncrono en la programación de sistemas, se refiere a la conexión o dependencia sobre otra tarea o proceso, es decir, cuando se ejecuta un proceso síncrono se espera a que termine dicha ejecución antes de continuar con otro proceso.

SYNCHRONOUS



Un proceso A se ejecuta y sólo cuando este finaliza, comienza la ejecución de un proceso B.

Tareas, procesos síncronos

Si declaramos constantes numéricas que indique el tiempo de una tarea, por ejemplo:

```
const val TIME_TO_COOK_IN_MINUTES: Int = 3;
const val TIME_TO_EAT_IN_MINUTES: Int = 2;
```

Un método que muestre los segundos que se toman dichas tareas expresadas en minutos:

```
fun getSecondsSpendByTask(TIME_OF_TASK: Int, tarea: String){
    println(tarea)
    for(minutes in 0..TIME_OF_TASK) {
        var segundosTranscurridos = minutes * 60;
        println("$segundosTranscurridos segundos")
        threadRealTime()
    }
}
```

Tareas, procesos síncronos

Otro método que haga la simulación del tiempo real con un thread, y que utilizamos en nuestra función anterior “getSecondsSpendByTask” de la siguiente forma:

```
fun threadRealTime(){  
    try{  
        Thread.sleep(60000)//60 segundos equivalentes a 1 minuto  
    }catch (ie: InterruptedException){  
        ie.printStackTrace()  
    }  
}
```

Nota: el método sleep() de la clase Thread recibe milisegundos como parámetros.

Tareas, procesos síncronos

Un método que represente el ejemplo real, de una persona que necesita cocinar para comer.

```
fun totalSynchronousTasksTimes(){  
    getSecondsSpendByTask(TIME_TO_COOK_IN_MINUTES, "COCINANDO")  
    getSecondsSpendByTask(TIME_TO_EAT_IN_MINUTES, "COMIENDO")  
}
```

Tareas, procesos síncronos

Al invocar este último método iniciaremos un proceso síncrono que ejecuta dos tareas, la primera cocinar y la segunda comer, demostrando que solo si terminamos de cocinar se podrá comenzar a comer. Para probar los métodos anteriores podemos crear el método main, dentro de una clase Utils que los contenga de la siguiente forma:

```
fun main(){  
    val utils = Utils()  
    try {  
        utils.totalSynchronousTasksTimes()  
    }catch (e: Exception){e.printStackTrace()}  
}
```


Tareas, procesos síncronos

Nuestra salida por consola en el terminal sería la siguiente:

```
"C:\Program Files\Android\Android Studio\jre\bin\java.exe" ...  
COCINANDO  
0 segundos  
60 segundos  
120 segundos  
180 segundos  
COMIENDO  
0 segundos  
60 segundos  
120 segundos  
  
Process finished with exit code 0
```

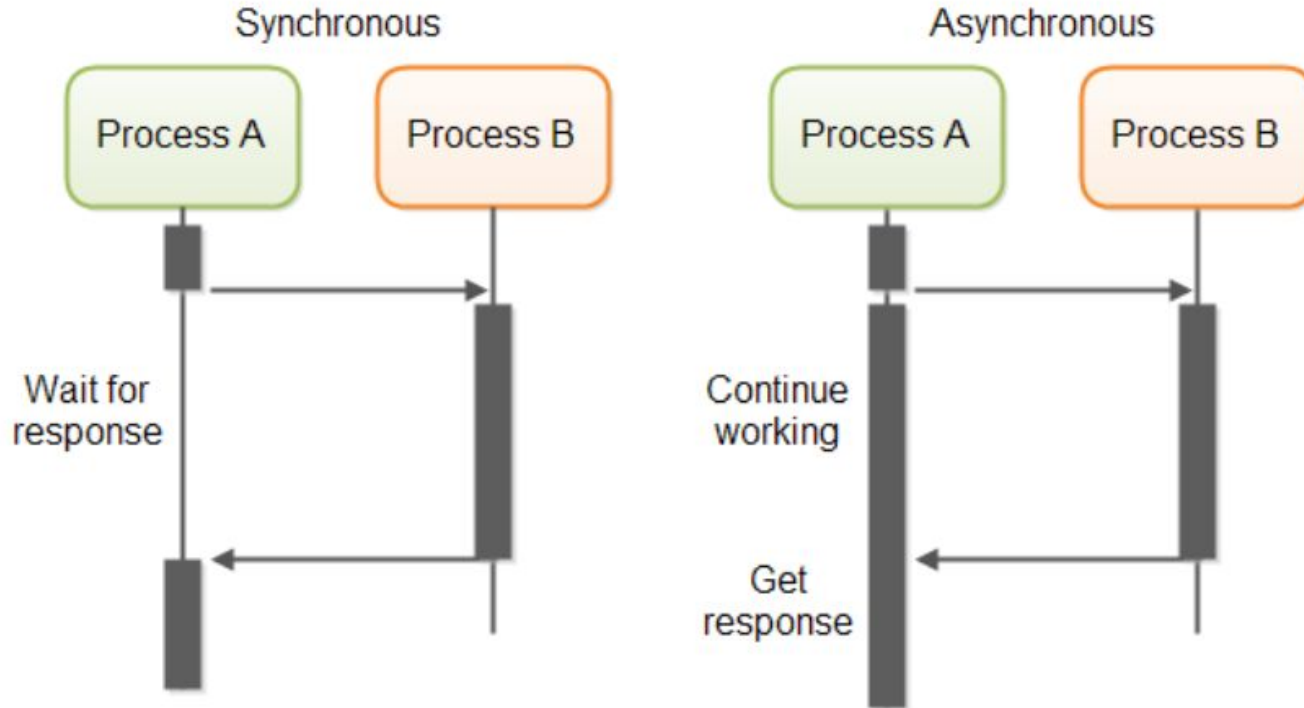
Tareas, procesos asíncronos

Una tarea o proceso asíncrono, en el modelo de programación de sistemas es aquel que se inicia, permitiendo continuar con otros procesos, para posteriormente recibir una respuesta; en otras palabras, cuando se ejecuta un proceso asíncrono, podemos iniciar otro proceso sin esperar a que el primero sea completado.

ASYNCHRONOUS



Tarea Síncrona vs Asíncrona



Tareas, procesos asíncronos

Para lograr constituir procesos asíncronos en android, disponemos de algunas técnicas para su correcto funcionamiento, las principales serían las siguientes:

- Threading
- Callbacks
- Promises
- Reactive Extensions (observable)
- Coroutines

Tareas, procesos asíncronos

Hagamos el ensayo de comer, cocinar y lavar creando procesos asíncronos independientes con un breve ejemplo, creamos una clase “UtilsAsync” donde copiaremos los siguientes métodos de nuestro ejemplo de tareas síncronas anterior:

```
fun getSecondsSpendByTask(TIME_OF_TASK: Int, tarea: String) {  
    for (minutes in 0..TIME_OF_TASK) {  
        println(tarea)  
        var segundosTranscurridos = minutes * 60;  
        println("$segundosTranscurridos segundos")  
        threadRealTime()  
    }  
}  
fun threadRealTime() {  
    try {  
        Thread.sleep(6000)  
    } catch (ie: InterruptedException) {  
        ie.printStackTrace()  
    }  
}
```

Tareas, procesos asíncronos

Del código anterior, destacamos el `println`, dado que para este ejemplo es mejor moverlo dentro del ciclo `for` del método `getSecondsSpendByTask()`.

Creamos las siguientes variables constantes:

```
const val TIME_TO_COOK_POTATOES_IN_MINUTES: Int = 3;
const val TIME_TO_EAT_BREAD_IN_MINUTES: Int = 2;
const val TIME_TO_WASH_CLOTHES_IN_MINUTES: Int = 4;
```

Tareas, procesos asíncronos

Y finalmente creamos un método que nos permita una ejecución independiente de las tres tareas a través de threads de la siguiente forma:

```
fun doAsyncTasks() {  
  
    Thread(Runnable {  
        getSecondsSpendByTask(TIME_TO_EAT_BREAD_IN_MINUTES, "COMIENDO")  
    }).start()  
  
    Thread(Runnable {  
        getSecondsSpendByTask(TIME_TO_COOK_POTATOES_IN_MINUTES, "COCINANDO PAPAS")  
    }).start()  
  
    Thread(Runnable {  
        getSecondsSpendByTask(TIME_TO_WASH_CLOTHES_IN_MINUTES, "LAVANDO ROPA")  
    }).start()  
}
```

Tareas, procesos asíncronos

Para probar el ensayo, copiamos nuestro método main del ejemplo anterior, indicando en esta ocasión nuestra nueva clase de la siguiente forma:

```
fun main(){  
    val utilsAsync = UtilsAsync()  
    try {  
        utilsAsync.doAsyncTasks()  
    }catch (e: Exception){e.printStackTrace()}  
}
```


Tareas, procesos asíncronos

El resultado por consola sería el siguiente:

```
"C:\Program Files\Android\Android Studio\jre\bin\ja
COCINANDO PAPAS
0 segundos
LAVANDO ROPA
0 segundos
COMIENDO
0 segundos
LAVANDO ROPA
COMIENDO
60 segundos
COCINANDO PAPAS
60 segundos
60 segundos
COCINANDO PAPAS
COMIENDO
120 segundos
120 segundos
LAVANDO ROPA
120 segundos
COCINANDO PAPAS
180 segundos
LAVANDO ROPA
180 segundos
LAVANDO ROPA
```

Procesos Background

El Hilo principal (Thread UI)

- manejar la interfaz de usuario
- coordinar las interacciones del usuario
- recibir eventos del ciclo de vida de la aplicación.

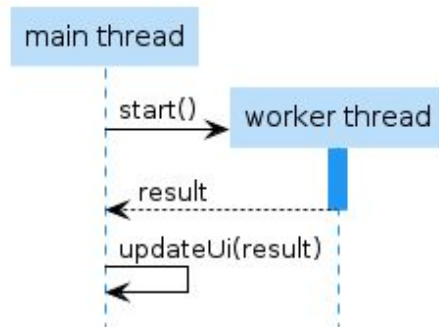
Background

- Evitan el bloqueo del hilo principal al realizar tareas largas
- Permiten realizar otras tareas por el usuario, mientras el subproceso se ejecuta

Desafíos de los procesos background

Las tareas en segundo plano consumen los recursos limitados de un dispositivo, como la RAM y la batería. Esto puede resultar en una mala experiencia para el usuario si no se maneja correctamente.

Para maximizar la batería y aplicar un buen comportamiento de la aplicación, Android restringe el trabajo en segundo plano cuando la aplicación (o una notificación de servicio en primer plano) no es visible para el usuario.



Proceso en el main thread UI

- Cuando el usuario inicia su aplicación, Android crea un nuevo proceso de Linux junto con un hilo de ejecución.
- Mientras visualizamos una animación o una actualización de la pantalla en nuestra aplicación, el sistema intenta ejecutar un bloque de trabajo sobre el hilo principal.
- Si el hilo principal no puede terminar de ejecutar bloques de trabajo en el tiempo que realiza la renderización, podemos observar bloqueos, retrasos

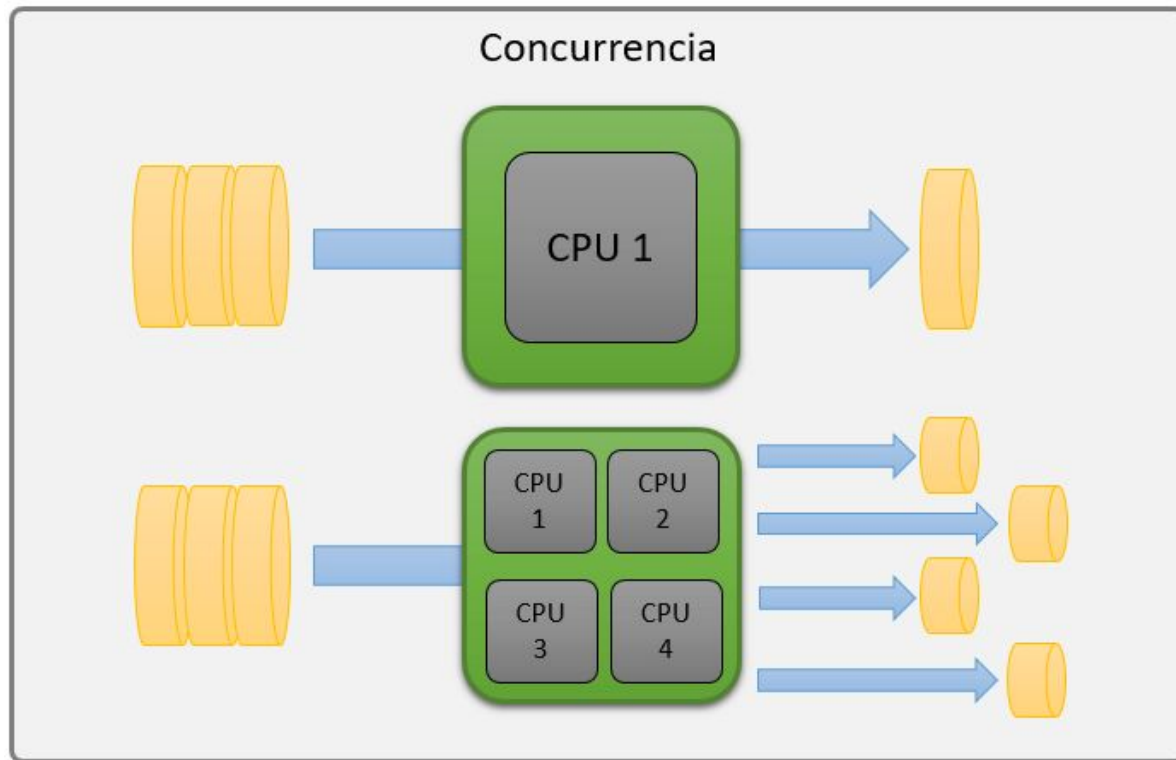
Concurrencia y Paralelismo

Concurrencia

La concurrencia es básicamente la capacidad de ejecutar múltiples tareas o procesos al mismo tiempo, pero no necesariamente simultáneos. En una aplicación concurrente tres procesos pueden iniciarse y completarse en períodos de tiempo distintos.

En principio, un dispositivo que dispone de un único microprocesador solamente puede ejecutar un proceso al mismo tiempo. No es posible ejecutar otro proceso hasta que ha finalizado el anterior.

Concurrencia



Context Switching

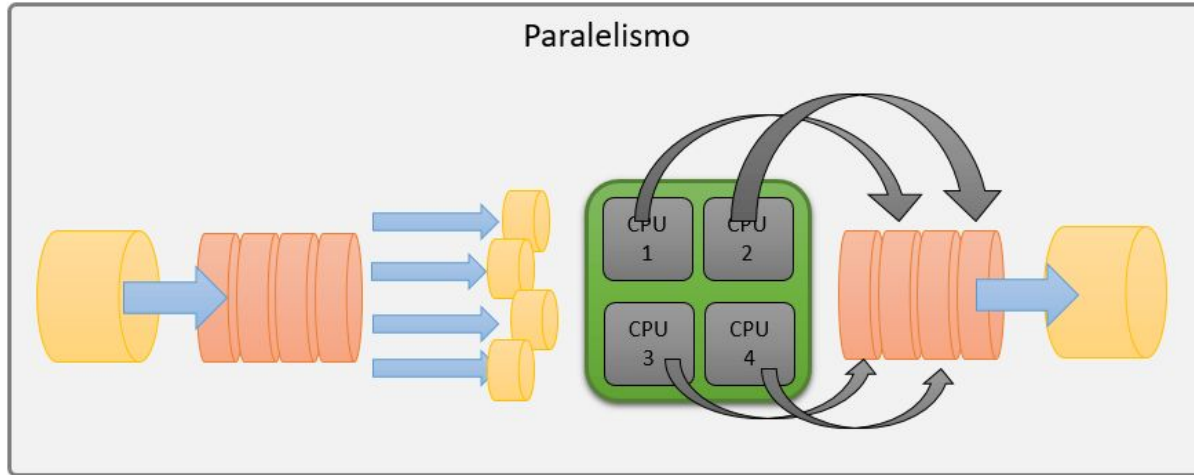
Un context-switching consiste en la ejecución de una rutina perteneciente al núcleo del sistema operativo multitarea de una computadora, cuyo propósito es parar la ejecución de un hilo o proceso para dar paso a la ejecución de otro distinto.

Es de esta manera que un procesador single-core ejecuta tareas multiproceso, asignando temporalmente el estado “en espera” a la ejecución de un subproceso dado, mientras inicia o retoma la ejecución de otro proceso.

Paralelismo

El paralelismo es un tipo de concurrencia que se distingue principalmente por la capacidad que tiene el CPU para trabajar más de un proceso al mismo tiempo. Un procesador puede trabajar con una cantidad de procesos igual a su cantidad de cores, es decir, si un procesador tiene un solo core, entonces solo podrá ejecutar un proceso a la vez, por otro parte, si tenemos un procesador QUAD-CORE (4), entonces podremos ejecutar hasta cuatro procesos al mismo tiempo.

Paralelismo



Threads

Procesos

Un proceso es una instancia de un programa en ejecución. Un programa puede tener múltiples procesos. Un proceso generalmente comienza con un solo subproceso, en android lo conocemos como main thread UI, es decir, un subproceso primario, sin embargo, en el camino del proceso principal se pueden crear múltiples subprocesos.

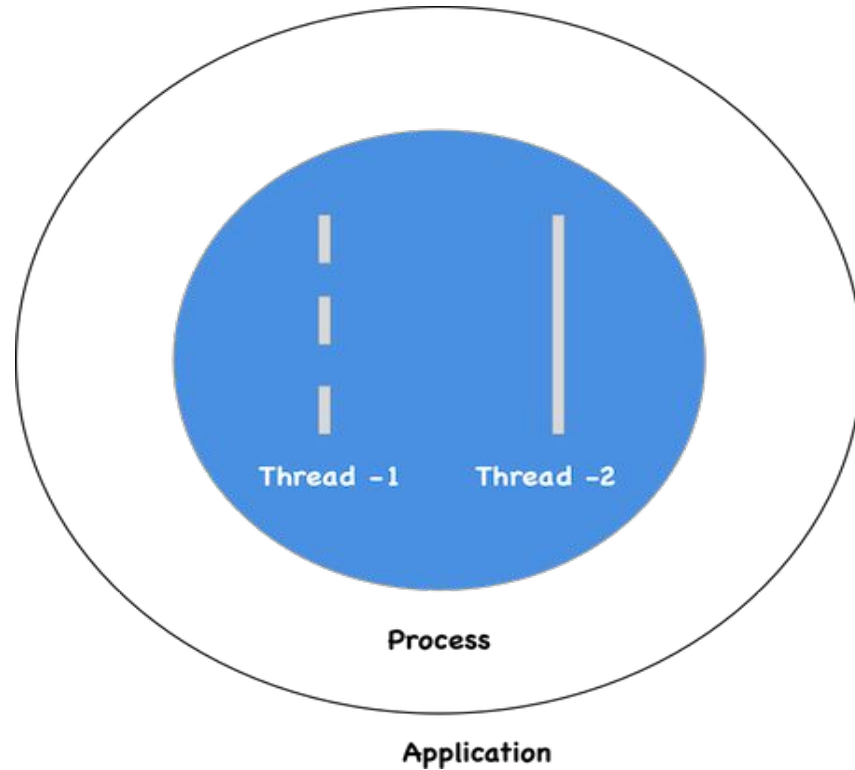
Threads

Los hilos (threads o subproceso) son una secuencia de ejecución de código que puede ejecutarse independientemente uno del otro. **Es la unidad de tareas más pequeña que puede ejecutar un sistema operativo.** Un programa puede ser de un solo subproceso o de subprocesos múltiples.

Los threads o hilos son la manera más tradicional dentro del mundo de la programación para evitar el bloqueo de aplicaciones.

Es fundamental para la capacidad de respuesta de la aplicación a través de su main thread IU, que no sea bloqueado por subprocesos largos y complejos.

Threads



Threads

El código siguiente muestra un gestor que descarga una imagen desde la web en un subproceso separado a través de thread, y la asigna a una vista de tipo ImageView:

```
public void onClick(View v) {  
    new Thread(new Runnable() {  
        public void run() {  
            Bitmap b = loadImageFromNetwork("http://example.com/image.png");  
            mImageView.setImageBitmap(b);  
        }  
    }).start();  
}
```

Threads

El ejemplo anterior, está correctamente diseñado para evitar sobrecargar el main thread; sin embargo, tenemos un detalle, android no aconseja interactuar con los objetos de vista desde un subproceso, como es el caso de asignar el bitmap al objeto mImageView. Para solucionarlo, utilizamos el método View.post(Runnable) de la siguiente forma:

```
public void onClick(View v) {  
    new Thread(new Runnable() {  
        public void run() {  
            final Bitmap bitmap =  
                loadImageFromNetwork("http://example.com/image.png");  
            mImageView.post(new Runnable() {  
                public void run() {  
                    mImageView.setImageBitmap(bitmap);  
                }  
            });  
        }  
    }).start();  
}
```


Main Thread Android

Al desarrollar aplicaciones de Android, siempre debemos ser conscientes de nuestro Main Thread.

El hilo principal está ocupado lidiando con cosas cotidianas, como dibujar nuestra interfaz de usuario, responder a las interacciones del usuario y, en general, de forma predeterminada, ejecutar en su mayoría el código que escribimos.

Un buen desarrollador sabe que necesita descargar tareas pesadas para un subproceso de trabajo para evitar obstruir el subproceso principal y permitir una experiencia de usuario más fluida y evitar el bloqueo de la aplicación con el mensaje “no responde la aplicación”. (error ANR, Application Not Responding).

Main Thread Android

Cuando llega el momento de actualizar la interfaz de usuario, debemos volver al hilo principal, ya que solo este puede tocar y actualizar la interfaz de usuario de la aplicación.

Una forma común de lograr esto es llamar al método `runOnUiThread ()` de la Actividad:

```
runOnUiThread(new Runnable() {  
    void run() {  
        // Do stuff...  
    }  
});
```

Threads Síncronos y Asíncronos

- Synchronous single thread:

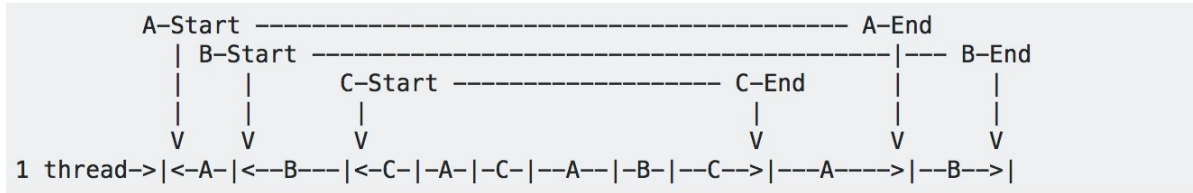
```
1 thread -> |<---A--->||<---B----->||<---C----->|
```

- Synchronous multi thread:

```
thread A -> |<---A--->|  
          \  
thread B -----> ->|<---B----->|  
                \  
thread C -----> ->|<---C----->|
```

Threads Síncronos y Asíncronos

- Asynchronous single thread:



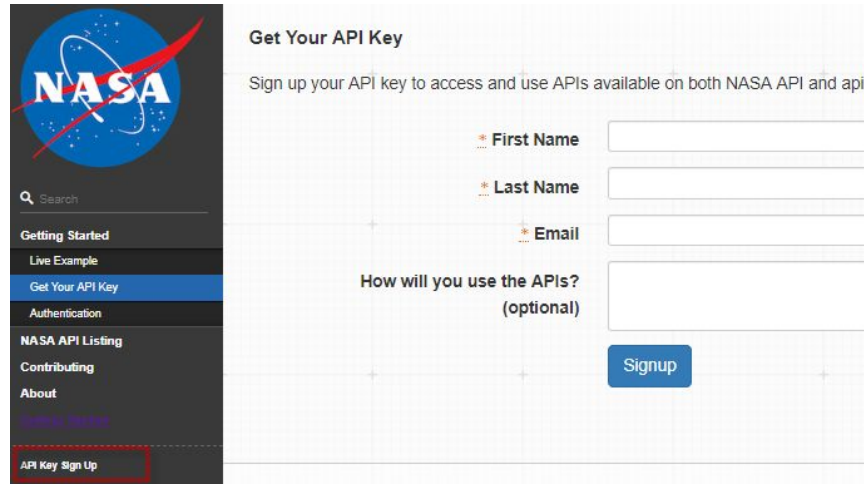
- Asynchronous multi thread:

```
thread A -> |<---A--->|  
thread B -----> |<---B----->|  
thread C -----> |<---C----->|
```

Ejercicio 1

Obtención de datos desde el api de la nasa para ser utilizados sus datos .

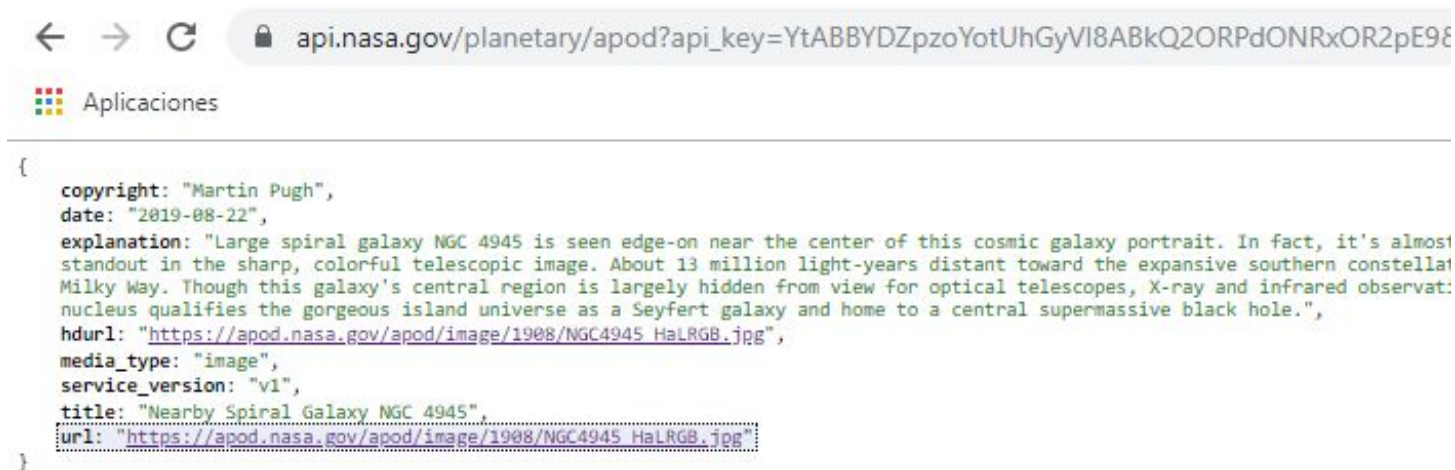
Utilizaremos imágenes de la nasa, por lo tanto la primera acción que realizaremos es ingresar a la página <http://api.nasa.gov> en la cuál crearemos nuestra llave propia para consumir un método que retorna la url de la imagen del día.



The image shows a screenshot of the NASA API Key Sign Up page. On the left is a dark sidebar with the NASA logo at the top. Below the logo is a search bar and a list of navigation links: 'Getting Started', 'Live Example', 'Get Your API Key' (highlighted in blue), 'Authentication', 'NASA API Listing', 'Contributing', 'About', and 'Contact Us'. At the bottom of the sidebar is a red dashed box around the 'API Key Sign Up' link. The main content area is white and titled 'Get Your API Key'. It contains the text 'Sign up your API key to access and use APIs available on both NASA API and api.' followed by three input fields for 'First Name', 'Last Name', and 'Email'. Below these is a larger text area for 'How will you use the APIs? (optional)'. A blue 'Signup' button is at the bottom right of the form.

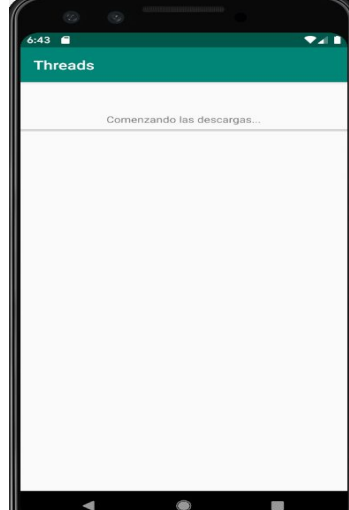
Ejercicio 1 - Solución

Una vez obtenida nuestra api key, vamos al navegador y utilizamos el método que retorna la imagen del día agregando nuestra api key en la url.

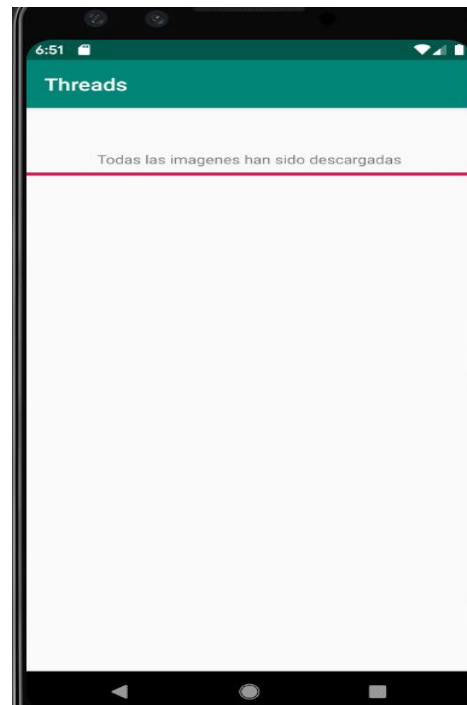
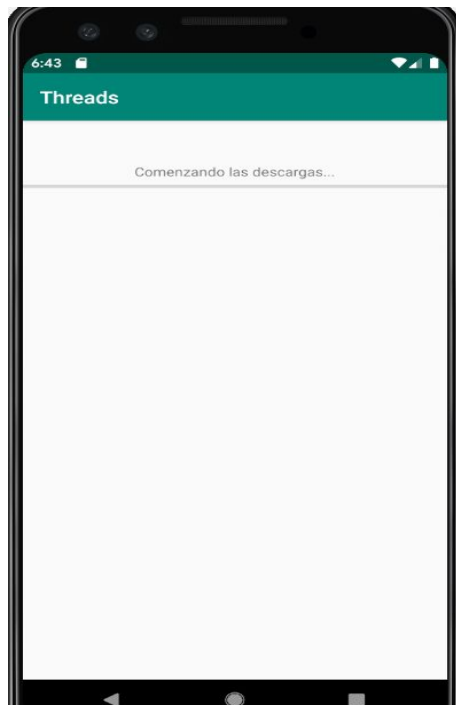


Ejercicio 2

Construir una aplicación que al iniciarse descargue una misma imagen en ocho(8) oportunidades a través de hilos simultáneos utilizando la clase `ThreadPoolExecutor` aprovechando la máxima capacidad posible del teléfono a través de sus múltiples cores. Dependiendo del teléfono podremos visualizar dos hilos, cuatro hilos hasta ocho hilos simultáneos en el Logcat. Se deberá mostrar un progressbar que indique el progreso de la descarga y también el inicio y fin de los hilos en los logs de la aplicación.



Ejercicio 2 - Solución



Ejercicio 2 - Solución

Logcat

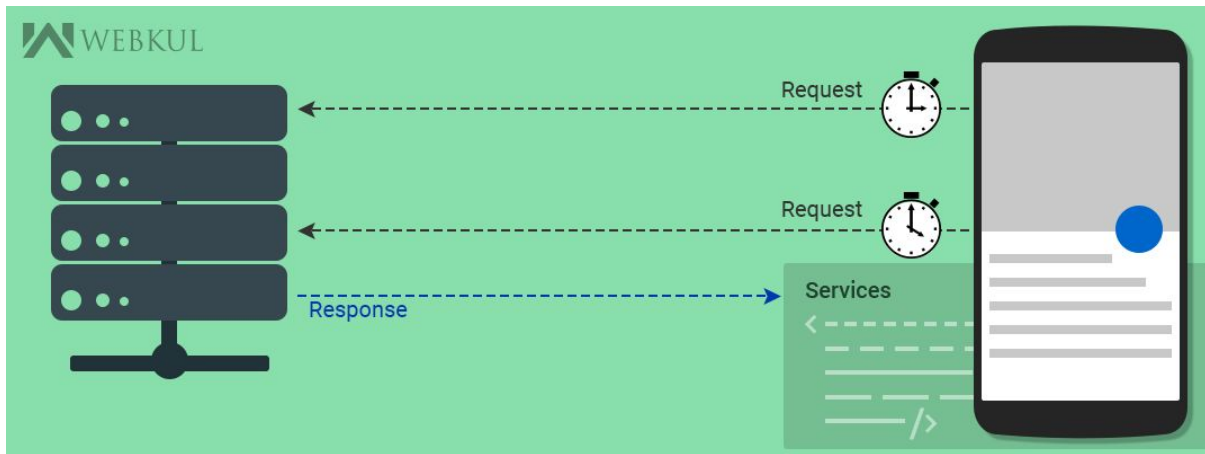
Emulador Pixel_3_API_29 Android cl.desafiolatam.threads (7378) Info DownloadThread

```
2019-08-28 15:45:36.222 7378-8186/cl.desafiolatam.threads I/DownloadThread: Iniciando Thread : 0
2019-08-28 15:45:36.264 7378-8187/cl.desafiolatam.threads I/DownloadThread: Iniciando Thread : 1
2019-08-28 15:45:36.274 7378-8188/cl.desafiolatam.threads I/DownloadThread: Iniciando Thread : 2
2019-08-28 15:45:36.277 7378-8190/cl.desafiolatam.threads I/DownloadThread: Iniciando Thread : 3
2019-08-28 15:45:42.157 7378-8188/cl.desafiolatam.threads I/DownloadThread: Thread completado 2
2019-08-28 15:45:42.158 7378-8188/cl.desafiolatam.threads I/DownloadThread: Iniciando Thread : 4
2019-08-28 15:45:42.404 7378-8187/cl.desafiolatam.threads I/DownloadThread: Thread completado 1
2019-08-28 15:45:42.405 7378-8187/cl.desafiolatam.threads I/DownloadThread: Iniciando Thread : 5
2019-08-28 15:45:42.549 7378-8190/cl.desafiolatam.threads I/DownloadThread: Thread completado 3
2019-08-28 15:45:42.551 7378-8190/cl.desafiolatam.threads I/DownloadThread: Iniciando Thread : 6
2019-08-28 15:45:42.556 7378-8186/cl.desafiolatam.threads I/DownloadThread: Thread completado 0
2019-08-28 15:45:42.558 7378-8186/cl.desafiolatam.threads I/DownloadThread: Iniciando Thread : 7
2019-08-28 15:45:44.466 7378-8186/cl.desafiolatam.threads I/DownloadThread: Thread completado 7
2019-08-28 15:45:44.821 7378-8188/cl.desafiolatam.threads I/DownloadThread: Thread completado 4
2019-08-28 15:45:45.087 7378-8187/cl.desafiolatam.threads I/DownloadThread: Thread completado 5
2019-08-28 15:45:45.339 7378-8190/cl.desafiolatam.threads I/DownloadThread: Thread completado 6
```

Services

Service

Un servicio es un componente de aplicación que puede realizar operaciones de larga ejecución en segundo plano y que no proporciona una interfaz de usuario.



Download Manager Service

El Download Manager es un servicio del sistema que maneja descargas HTTP de larga duración. Los clientes pueden solicitar que se descargue un URI a un archivo de destino particular. El Download Manager realizará la descarga en segundo plano, se ocupará de las interacciones HTTP y volverá a intentar las descargas después de fallas o en los cambios de conectividad y reinicios del sistema.

Iniciar - Enlazar un Service

Un servicio está "iniciado" cuando un componente de aplicación (como una actividad) lo inicia llamando a `startService()`.

Un servicio es de "de enlace" cuando un componente de la aplicación se vincula a él llamando al método `bindService()`.

Declarar un Service

Al igual que las actividades (y otros componentes), debes declarar todos los servicios en el archivo de manifiesto de tu aplicación.

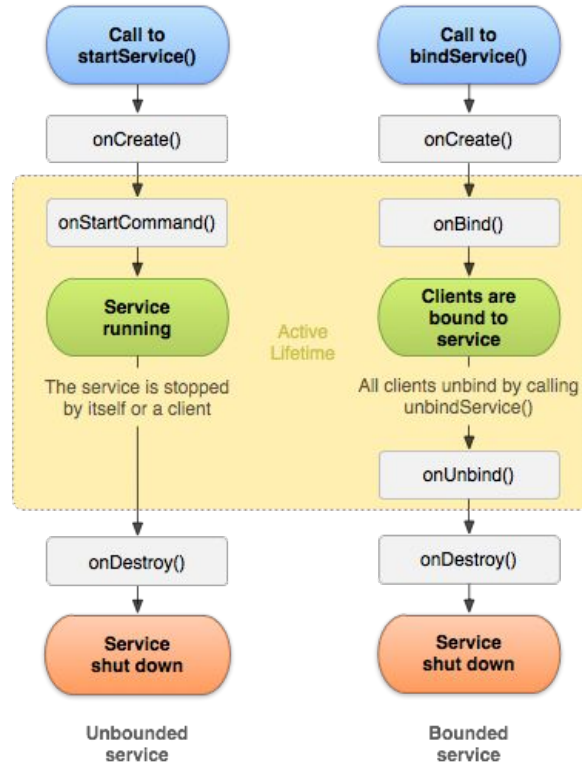
Para declarar tu servicio, agrega un elemento `<service>` como campo secundario del elemento de `<application>`.

```
<manifest ... >
  ...
  <application ... >
    <service android:name=".ExampleService" />
    ...
  </application>
</manifest>
```

Intent Service

- Crea un subproceso de trabajo predeterminado que ejecute todas las intents enviadas a `onStartCommand()`, independientemente del subproceso principal de tu aplicación.
- Crea una cola de trabajo que pase una intent a la vez a una implementación `onHandleIntent()`.
- Detiene el servicio una vez que se han manejado todas las solicitudes de inicio, para no tener que llamar al método `stopSelf()`.
- Proporciona la implementación predeterminada de `onBind()` que devuelve `NULL`.
- Proporciona una implementación predeterminada de `onStartCommand()` que envía el `intent` a la cola de trabajo y luego a la implementación del método `onHandleIntent()`.

Ciclo de Vida de un Service



Ejercicio 3

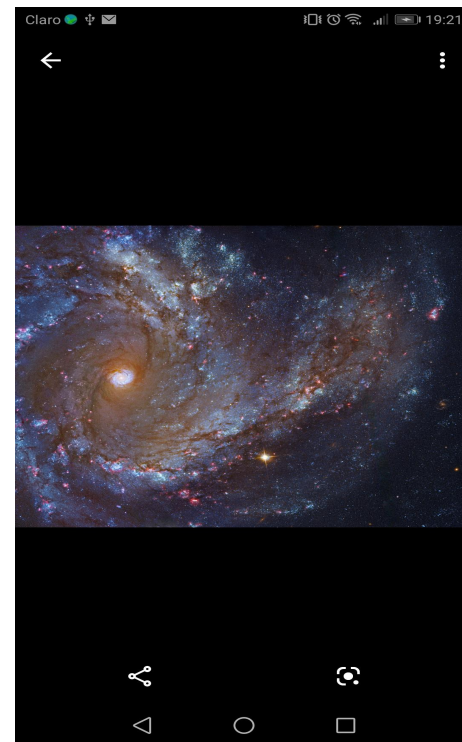
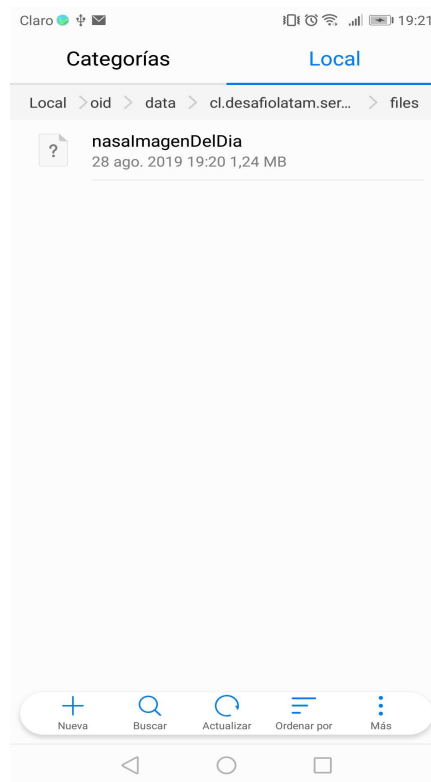
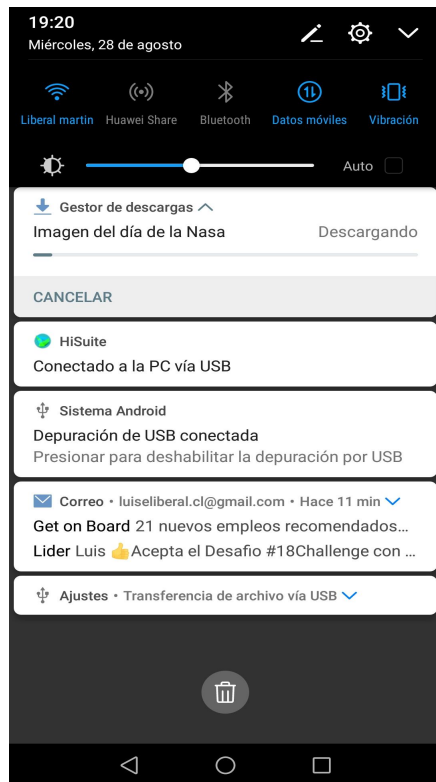
Crear un proyecto de nombre “ServiceDownloader” que utilice el api de la nasa para descargar la imagen del día en formato HD, a través de un proceso background utilizando la clase Download Manager de android.



Ejercicio 3 - Solución



{desafío}
latam_

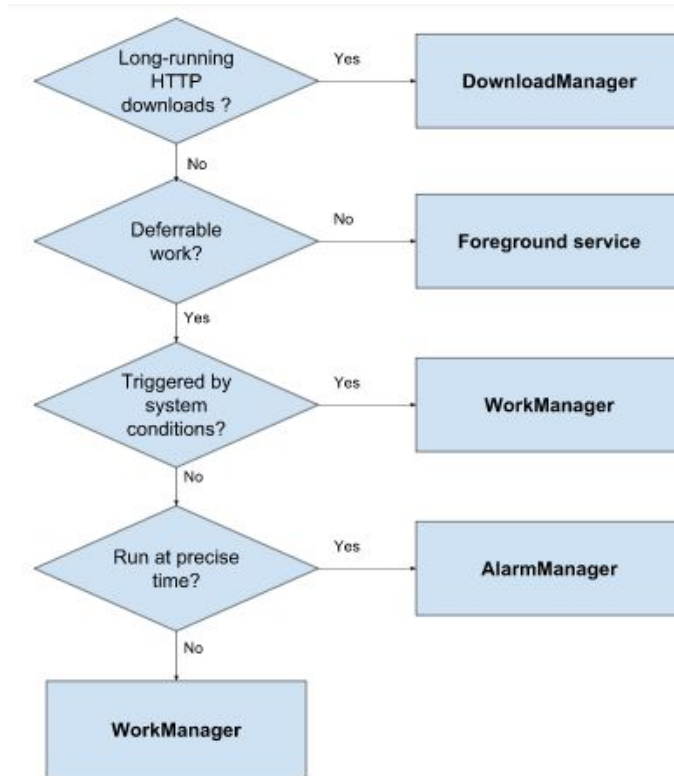


AlarmManager

AlarmManager

- Proporciona acceso a los servicios de alarma del sistema operativo android.
- Los servicios que habilita permiten programar tareas para que se ejecuten en algún momento del futuro.
- El AlarmManager mantiene un bloqueo de activación de la CPU siempre que el método del receptor de alarma se esté ejecutando.
- Garantiza que el teléfono no se suspenda hasta que haya terminado de manejar la tarea programada.
- Alarm Manager libera el bloqueo de activación una vez recibida la notificación.

Situación de uso del AlarmManager



Características del AlarmManager

- Permite ejecutar una tarea en un tiempo específico.
- Te permite disparar Intents en tiempos programados o intervalos de tiempo programados.
- Puedes usar AlarmManager junto con el componente BroadcastReceiver para iniciar servicios o ejecutar otras operaciones.
- Funcionan fuera de la aplicación, lo que te permite activar acciones o eventos fuera de tu aplicación sin necesidad de que esté funcionando.
- AlarmManager ayuda a minimizar los requerimientos de recursos en nuestras aplicaciones, sin necesidad de tener procesos background continuamente escuchando o corriendo.

Tipos de Alarma

- **ELAPSED_REALTIME**: activa el PendingIntent en función del tiempo transcurrido desde que se inició el dispositivo, pero no activa el dispositivo. El tiempo transcurrido incluye cualquier momento durante el cual el dispositivo estuvo dormido. Ejemplo:
- **ELAPSED_REALTIME_WAKEUP**: despierta el dispositivo y dispara el PendingIntent después de que haya transcurrido el tiempo especificado desde el inicio del dispositivo.

```
// Hopefully your alarm will have a lower frequency than this!  
alarmMgr?.setInexactRepeating(  
    AlarmManager.ELAPSED_REALTIME_WAKEUP,  
    SystemClock.elapsedRealtime() + AlarmManager.INTERVAL_HALF_HOUR,  
    AlarmManager.INTERVAL_HALF_HOUR,  
    alarmIntent  
)
```

Tipos de Alarma

- **RTC**: activa el PendingIntent a la hora especificada pero no activa el dispositivo.
- **RTC_WAKEUP**: despierta el dispositivo para disparar el PendingIntent a la hora especificada.

Ejemplo:

```
// Set the alarm to start at approximately 2:00 p.m.
val calendar: Calendar = Calendar.getInstance().apply {
    timeInMillis = System.currentTimeMillis()
    set(Calendar.HOUR_OF_DAY, 14)
}

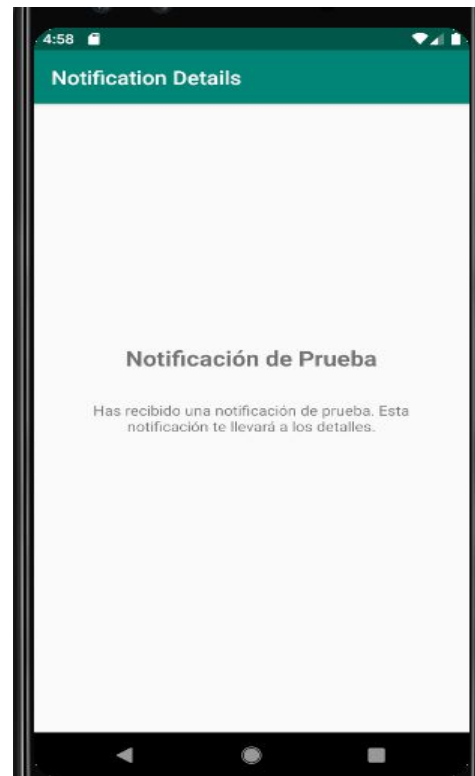
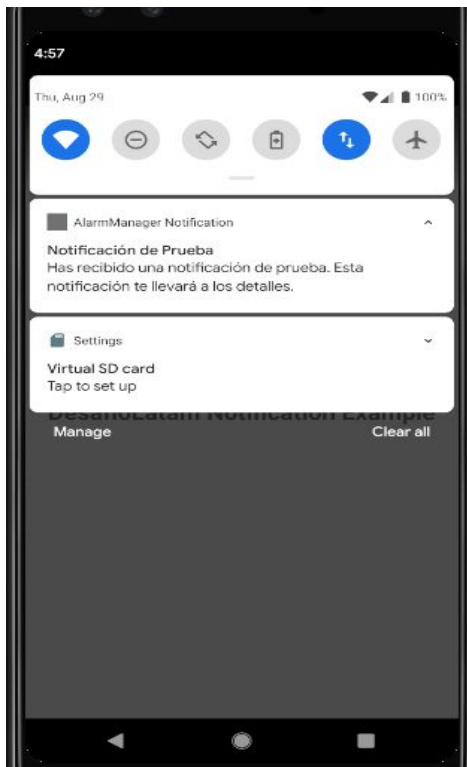
// With setInexactRepeating(), you have to use one of the AlarmManager interval
// constants--in this case, AlarmManager.INTERVAL_DAY.
alarmMgr?.setInexactRepeating(
    AlarmManager.RTC_WAKEUP,
    calendar.timeInMillis,
    AlarmManager.INTERVAL_DAY,
    alarmIntent
)
```


Ejercicio 4

Crear un proyecto de nombre “AlarmManagerNotification” que muestre un título centrado con el mensaje “Desafío Latam Notification Example” en su actividad principal y ejecute una notificación a los 10 segundos de iniciarse la aplicación utilizando las clases AlarmManager, AlarmReceiver, BroadcastReceiver, Notification, NotificationManager, IntentService.



Ejercicio 4 - Solución



WorkManager

Introducción

En este capítulo estudiaremos la creación de tareas programables en android con la biblioteca WorkManager, la cual garantiza la ejecución óptima de los procesos agendados en segundo plano. WorkManager es una de las librerías más modernas para crear tareas desarrollada por android.

Las tareas programables son parte esencial de la actividad laboral, como desarrolladores en ocasiones nos solicitarán ejecutar algún tipo de operación repetitiva y a determinado horario, es en este tipo de ocasiones donde nos será muy útil aprender a utilizar WorkManager.

Objetivos

- Conocer la agenda de tareas con WorkManager en android
- Implementar WorkManager en un proyecto android

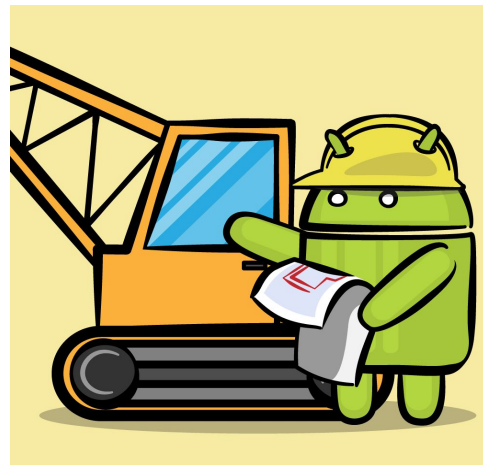


WorkManager

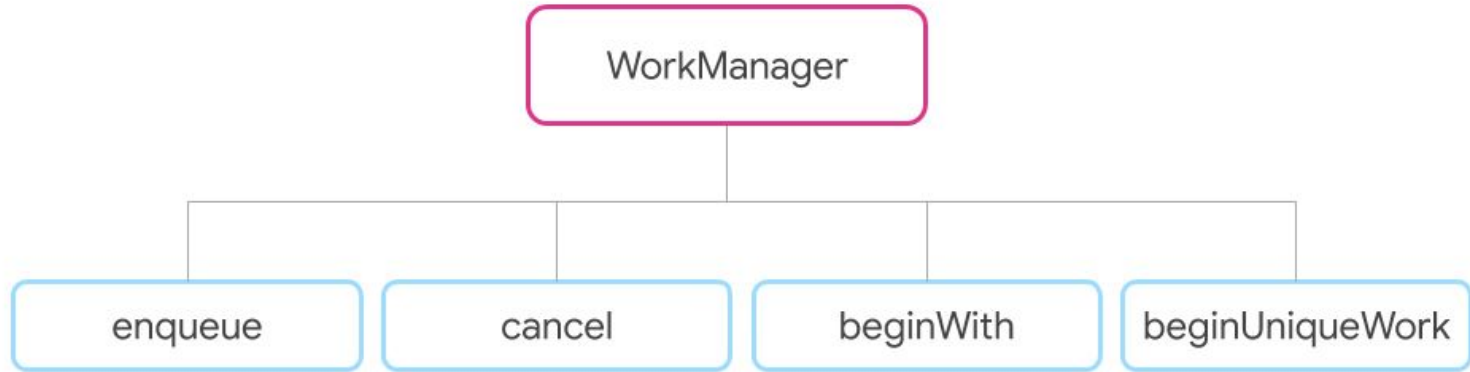
WorkManager es una biblioteca de Android que ejecuta trabajos en segundo plano diferibles cuando se cumplen las restricciones del trabajo. Está destinado a tareas que requieren garantizar que el sistema los ejecutará incluso si la aplicación sales.

WorkManager proporciona una API amigable con la batería que encapsula años de evolución de las restricciones de comportamiento en segundo plano de Android.

```
implementation "android.arch.work:work-runtime-ktx:1.0.0-beta01"  
androidTestImplementation  
"android.arch.work:work-testing:1.0.0-beta01"
```

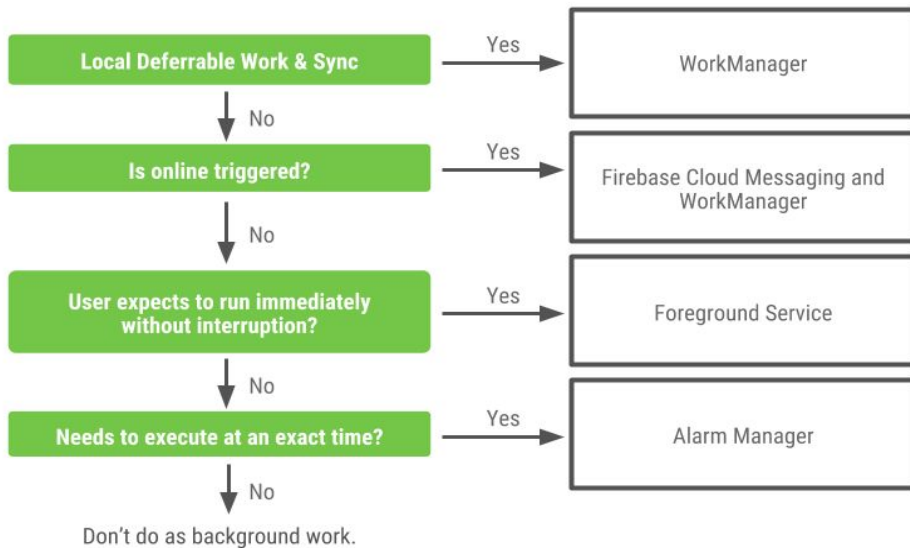


Principales métodos de WorkManager



Evaluando el uso de WorkManager

I need to run a task in background, how should I do it?



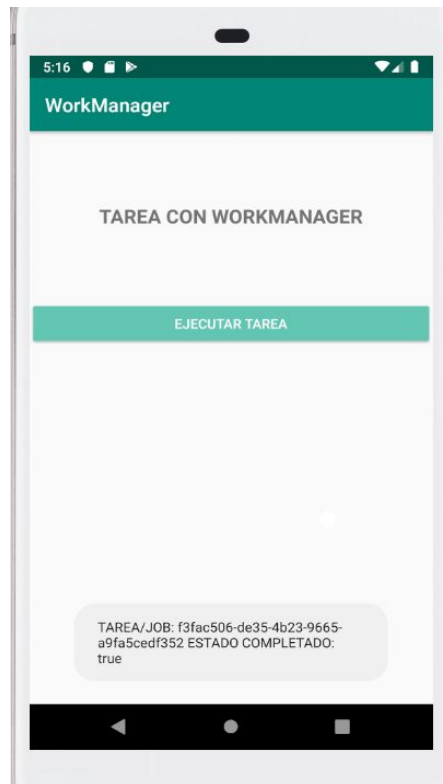
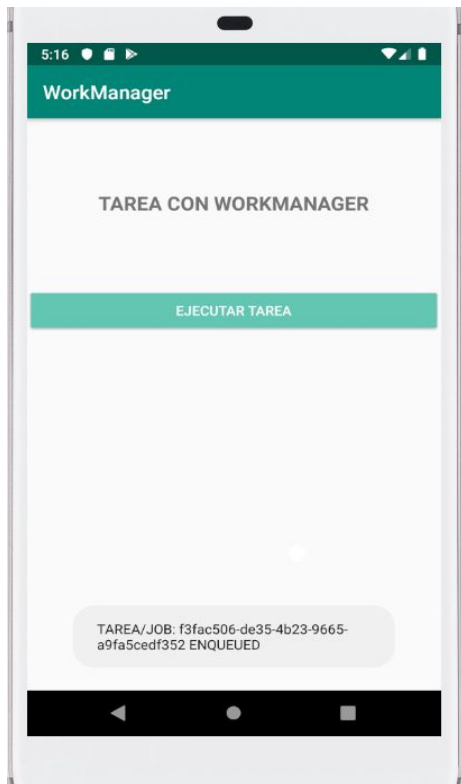
Beneficios de utilizar WorkManager

- Maneja la compatibilidad con diferentes versiones del sistema operativo
- Sigue las mejores prácticas de salud del sistema
- Admite tareas asincrónicas únicas y periódicas.
- Admite tareas encadenadas con entrada / salida
- Le permite establecer restricciones cuando se ejecuta la tarea
- Garantiza la ejecución de tareas, incluso si la aplicación o el dispositivo se reinicia.

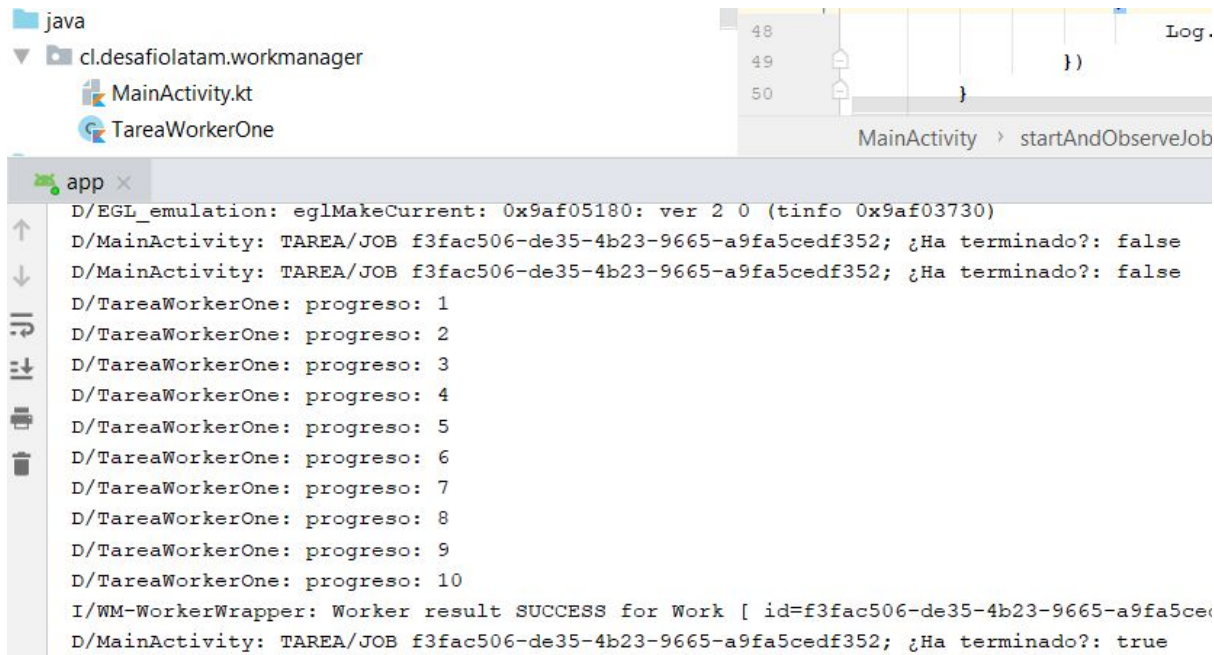
Ejercicio 1

Crear una aplicación que realice un trabajo en segundo plano utilizando la librería WorkManager. La tarea a realizar será contar del 1 al 10, haciendo esperar un segundo al sistema por cada número (cuenta real). La app ha de informar por medio de un mensaje Toast cuando se inicia la tarea y cuando se termina, mostrando también por consola o log la ejecución de la tarea.

Ejercicio 1 - Solución



Ejercicio 1 - Solución



The screenshot shows the Android Studio interface. On the left, the 'Project' view displays the file structure: a 'java' package containing 'cl.desafiolatam.workmanager', which includes 'MainActivity.kt' and 'TareaWorkerOne'. On the right, the 'Logcat' window shows the following log messages:

```
Log.  
48  
49  
50  
MainActivity > startAndObserveJob  
D/EGL_emulation: eglMakeCurrent: 0x9af05180: ver 2 0 (tinfo 0x9af03730)  
D/MainActivity: TAREA/JOB f3fac506-de35-4b23-9665-a9fa5cedf352; ¿Ha terminado?: false  
D/MainActivity: TAREA/JOB f3fac506-de35-4b23-9665-a9fa5cedf352; ¿Ha terminado?: false  
D/TareaWorkerOne: progreso: 1  
D/TareaWorkerOne: progreso: 2  
D/TareaWorkerOne: progreso: 3  
D/TareaWorkerOne: progreso: 4  
D/TareaWorkerOne: progreso: 5  
D/TareaWorkerOne: progreso: 6  
D/TareaWorkerOne: progreso: 7  
D/TareaWorkerOne: progreso: 8  
D/TareaWorkerOne: progreso: 9  
D/TareaWorkerOne: progreso: 10  
I/WM-WorkerWrapper: Worker result SUCCESS for Work [ id=f3fac506-de35-4b23-9665-a9fa5ce  
D/MainActivity: TAREA/JOB f3fac506-de35-4b23-9665-a9fa5cedf352; ¿Ha terminado?: true
```

{desafío}
latam_

*Academia de
talentos digitales*

www.desafiolatam.com