Guía de ejercicios - Programación asíncrona en Android (I)

 ¡Hola! Te damos la bienvenida a esta nueva guía de estudio.

¿En qué consiste esta guía?

La siguiente guía de estudio tiene como objetivo practicar y ejercitar los contenidos que hemos visto en clase.

¡Vamos con todo!

Tabla de contenidos

¡Comencemos!

Actividad guiada N°1: Android Background Service y HttpRequest paso a paso

En esta Actividad guiada usaremos un background service para hacer una petición http, este servicio se ejecutará al momento de iniciar el MainActivity y se detendrá al momento de detener la app.

¡Manos a la obra!

1. Creamos un proyecto nuevo con una Activity vacía.
2. Luego, creamos un paquete al cual llamaremos "`service`", el cual contendrá la clase del servicio.
3. A continuación, dentro de "`service`" creamos una clase nueva, la cual llamaremos `BackgroundService` y que extenderá de `Service()`
4. En la clase `BackgroundService`, necesitamos sobreescribir los siguientes métodos:
    a. **`onBind`**
    b. **`onCreate`**
    c. **`onDestroy`**
    d. **`onStartCommand`**
5. Ya que no estamos creando un Bound Service, podemos retornar null en el método **`onBind`**, quedando como por ejemplo:

```
override fun onBind(p0: Intent?): IBinder? = null
```

6. Antes de continuar debemos crear una inner class la cual ejecutará la tarea dentro del servicio cuando este sea llamado.

```
inner class HandleService(looper: Looper) : Handler(looper) {
        override fun handleMessage(msg: Message) {
            try {
                thread {
                    println("dummy request: ${dummyHttpRequest()}")
                }
            } catch (e: InterruptedException) {
                Thread.currentThread().interrupt()
            }
        }
    }
```

7. Ahora ya podemos sobreescribir el método `onCreate()`

```
override fun onCreate() {
    super.onCreate()
    HandlerThread("BackgroundService",
    Process.THREAD_PRIORITY_BACKGROUND).apply {
        start()
        serviceLooper = looper
        serviceHandler = HandleService(looper)
    }
}
```

8. En onDestroy llamaremos un Toast al momento de detener por completo la app, lo que a su vez detendrá el servicio.

```
override fun onDestroy() {
  super.onDestroy()
   Toast.makeText(this, "Deteniendo servicio", Toast.LENGTH_LONG).show()
}
```

9. Luego en onStartCommand llamaremos la clase `HandleService` la cual se definió en `onCreate()`

```
override fun onStartCommand(intent: Intent?, flags: Int, startId: Int):
Int {
  Toast.makeText(this, "Iniciando descarga", Toast.LENGTH_LONG).show()
  serviceHandler?.obtainMessage()?.also { message: Message ->
    message.arg1 = startId
    serviceHandler?.sendMessage(message)
    }
  return START_STICKY
}
```

10. Finalmente, tenemos que crear una función la cual ejecutara el http request.

```
private fun dummyHttpRequest(): String? {
        val url = URL("https://dummyjson.com/products/1")
        return (url.openConnection() as? HttpURLConnection)?.run {
            requestMethod = "GET"
            inputStream.bufferedReader().readText()
        }
    }
```

11. Ejecuta la aplicación y con la ayuda de Logcat podrás ver como después de iniciar la app y ejecutar el servicio, se hace el http request.

12. A continuación podemos ver todo el código del servicio:

```
class BackgroundService : Service() {

    private var serviceLooper: Looper? = null
    private var serviceHandler: HandleService? = null

    override fun onBind(p0: Intent?): IBinder? = null

    override fun onCreate() {
        super.onCreate()
                                        HandlerThread("BackgroundService",
Process.THREAD_PRIORITY_BACKGROUND).apply {
            start()
            serviceLooper = looper
            serviceHandler = HandleService(looper)
        }
    }

    override fun onDestroy() {
        super.onDestroy()
                          Toast.makeText(this,   "Deteniendo   servicio",
Toast.LENGTH_LONG).show()
    }

      override fun onStartCommand(intent: Intent?, flags: Int, startId:
Int): Int {
                           Toast.makeText(this,   "Iniciando   descarga",
Toast.LENGTH_LONG).show()
        serviceHandler?.obtainMessage()?.also { message: Message ->
            message.arg1 = startId
            serviceHandler?.sendMessage(message)
        }
        return START_STICKY
    }

    private fun dummyHttpRequest(): String? {
        val url = URL("https://dummyjson.com/products/1")
        return (url.openConnection() as? HttpURLConnection)?.run {
            requestMethod = "GET"
```

```
            inputStream.bufferedReader().readText()
        }
    }

    inner class HandleService(looper: Looper) : Handler(looper) {
        override fun handleMessage(msg: Message) {
            try {
                thread {
                    println("dummy request: ${dummyHttpRequest()}")
                }
            } catch (e: InterruptedException) {
                Thread.currentThread().interrupt()
            }
        }
    }
}
```

Actividad guiada N°2: Android Foreground Service paso a paso

En esta Actividad Guiada crearemos un Foreground Service, el cual se ejecutará al momento de abrir la app e inmediatamente mostrará una notificación la cual nos permitirá cerrar la app y seguir ejecutado el servicio.

¡Manos a la obra!

1. Creamos un proyecto nuevo con una Activity vacía.

2. Luego, creamos un paquete al cual llamaremos "`service`", el cual contendrá la clase del servicio.

3. A continuación, dentro de "`service`" creamos una clase nueva, la cual llamaremos `ForegroundService` y que extenderá de `Service()`

4. En la clase `ForegroundService`, necesitamos sobreescribir los siguientes métodos:
   - **`onBind`**
   - **`onCreate`**
   - **`onDestroy`**
   - **`onStartCommand`**

5. Ya que no estamos creando un Bound Service, podemos retornar null en el método **`onBind`**, quedando de la siguiente manera:

```
override fun onBind(p0: Intent?): IBinder? = null
```

6. Para iniciar y detener el servicio, podemos usar un compasión object el cual contendrá dos funciones: startService y stopService.

```
fun startService(context: Context, message: String) {
    ContextCompat.startForegroundService(
        context,
        Intent(context, ForegroundService::class.java).apply {
            putExtra("inputExtra", message)
        }
    )
}
```

```
fun stopService(context: Context) {
    context.stopService(Intent(context, ForegroundService::class.java))
}
```

¿Qué es lo que hacen estas funciones?

- **startService**: recibe como parámetros context y message, estos parámetros se pasan desde el MainActivity al momento de iniciar el servicio.
- **stopService**: recibe como parámetro context

7. Luego, en los métodos **onCreate** y **onDestroy**, crearemos un par de `Toast`, los cuales mostrarán un mensaje cuando el servicio se ejecute o se detenga respectivamente:

```
override fun onCreate() {
  super.onCreate()
  Toast.makeText(this, "Iniciando servicio", Toast.LENGTH_LONG).show()
}

override fun onDestroy() {
  super.onDestroy()
  Toast.makeText(this, "Deteniendo servicio", Toast.LENGTH_LONG).show()
}
```

8. Lo siguiente es crear una función la cual llamaremos dentro de `onStartCommand`, llamaremos a esta función `createNotificationChannel`

```
private fun createNotificationChannel() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        val serviceChannel = NotificationChannel(
           CHANNEL_ID, "Foreground Service Channel",
           NotificationManager.IMPORTANCE_DEFAULT
       )
    val manager = getSystemService(NotificationManager::class.java)
    manager.createNotificationChannel(serviceChannel)
    }
}
```

9. Ahora dentro de onStartCommand tenemos que recibir el string que definimos en el companion object en la función **startService**. Ya que es un Intent, lo haremos usando **getStringExtra** luego, con **NotificationCompat.Builder**, crearemos la

notificación que permitirá cerrar la app y continuar ejecutando el servicio. Finalmente, debemos retornar **START_NOT_STICKY** para especificar que tipo de servicio estamos ejecutando.

```
override fun onStartCommand(
    intent: Intent?,
    flags: Int,
    startId: Int
): Int {
    val input = intent?.getStringExtra("inputExtra")
    createNotificationChannel()
    val notificationIntent = Intent(this, MainActivity::class.java)
    val pendingIntent = PendingIntent.getActivity(
        this,
        0, notificationIntent, PendingIntent.FLAG_IMMUTABLE
    )

    val notification = NotificationCompat.Builder(this, CHANNEL_ID)
       .setContentTitle("Foreground Service")
       .setContentText(input)
       .setSmallIcon(android.R.drawable.ic_dialog_info)
       .setContentIntent(pendingIntent)
       .build()

    startForeground(1, notification)
    return START_NOT_STICKY
}
```

10. Finalmente, necesitamos llamar el servicio cuando se inicia MainActivity y detener el servicio cuando se llame al método onDestroy dentro del MainActivity.

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        ForegroundService.startService(this, "Ejecutando servicio")
    }

    override fun onDestroy() {
        super.onDestroy()
        ForegroundService.stopService(this)
    }
```

```
}
```

11. Ejecuta la app y ve el resultado, pon atención a que pasa cuando inicias la app, cuando te cambias a otra app y cuando cierras la app.

12. A continuación puedes ver el código del servicio completo:

```
class ForegroundService : Service() {

  private val CHANNEL_ID = "Foreground Service"

  override fun onBind(p0: Intent?): IBinder? = null

  override fun onCreate() {
    super.onCreate()
    Toast.makeText(this, "Iniciando servicio", Toast.LENGTH_LONG).show()
  }

  override fun onDestroy() {
   super.onDestroy()
   Toast.makeText(this, "Deteniendo servicio", Toast.LENGTH_LONG).show()
  }

 override fun onStartCommand(intent: Intent?, flags: Int, startId: Int):
Int {
  val input = intent?.getStringExtra("inputExtra")
  createNotificationChannel()
  val notificationIntent = Intent(this, MainActivity::class.java)
  val pendingIntent = PendingIntent.getActivity(
  this,0, notificationIntent,
  PendingIntent.FLAG_IMMUTABLE)

  val notification = NotificationCompat.Builder(this, CHANNEL_ID)
    .setContentTitle("Foreground Service")
    .setContentText(input)
    .setSmallIcon(android.R.drawable.ic_dialog_info)
    .setContentIntent(pendingIntent)
    .build()

    startForeground(1, notification)
    return START_NOT_STICKY
  }
```

```
  private fun createNotificationChannel() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
      val serviceChannel = NotificationChannel(
      CHANNEL_ID, "Foreground Service Channel",
      NotificationManager.IMPORTANCE_DEFAULT)
      val manager = getSystemService(NotificationManager::class.java)
      manager.createNotificationChannel(serviceChannel)
    }
  }

  companion object {
    fun startService(context: Context, message: String) {
      ContextCompat.startForegroundService(
        context,
          Intent(context, ForegroundService::class.java).apply {
            putExtra("inputExtra", message)
           }
         )
      }

  fun stopService(context: Context) {
    context.stopService(Intent(context, ForegroundService::class.java))
      }
   }
}
```

Preguntas de proceso

Reflexiona:

- A partir de lo aprendido hasta aquí ¿Qué contenido se me ha hecho más difícil? ¿Y cuál más sencillo?
- ¿Existe algún ejercicio de los resueltos previamente que deba repetir o desarrollar otra vez para poder dominarlo mejor?
- Nombra al menos un uso que pueda darle a lo aprendido hasta ahora.

Preguntas de cierre

- Nombra al menos 3 tareas que solo se deban ejecutar en el background thread.
- Compara el ciclo de vida de las Activities y los Services y revisa las similitudes.
- Revisa la documentación y responde, ¿por qué necesitamos crear esa inner class?
- ¿Qué pasa si ejecutamos una tarea en el Main Thread y toma más de 5 segundos responder?

Referencias bibliográficas

- Kotlin, documentación oficial:
https://kotlinlang.org/api/kotlinx.coroutines/kotlinx-coroutines-core/kotlinx.coroutines/run-blocking.html

- Android, documentación oficial:
https://developer.android.com/guide/components/services