La Actividad en Android (Activity)

1.Las Actividades •2.Ciclo de vida de una aplicación •3.Conservar el estado de una aplicación. •4.Intent y filtros •4.1.Intent Explícitos •4.2.Intent Implícitos •4.3.Elementos •4.4.Propagación •4.5.Filtros •4.6.PendingIntent

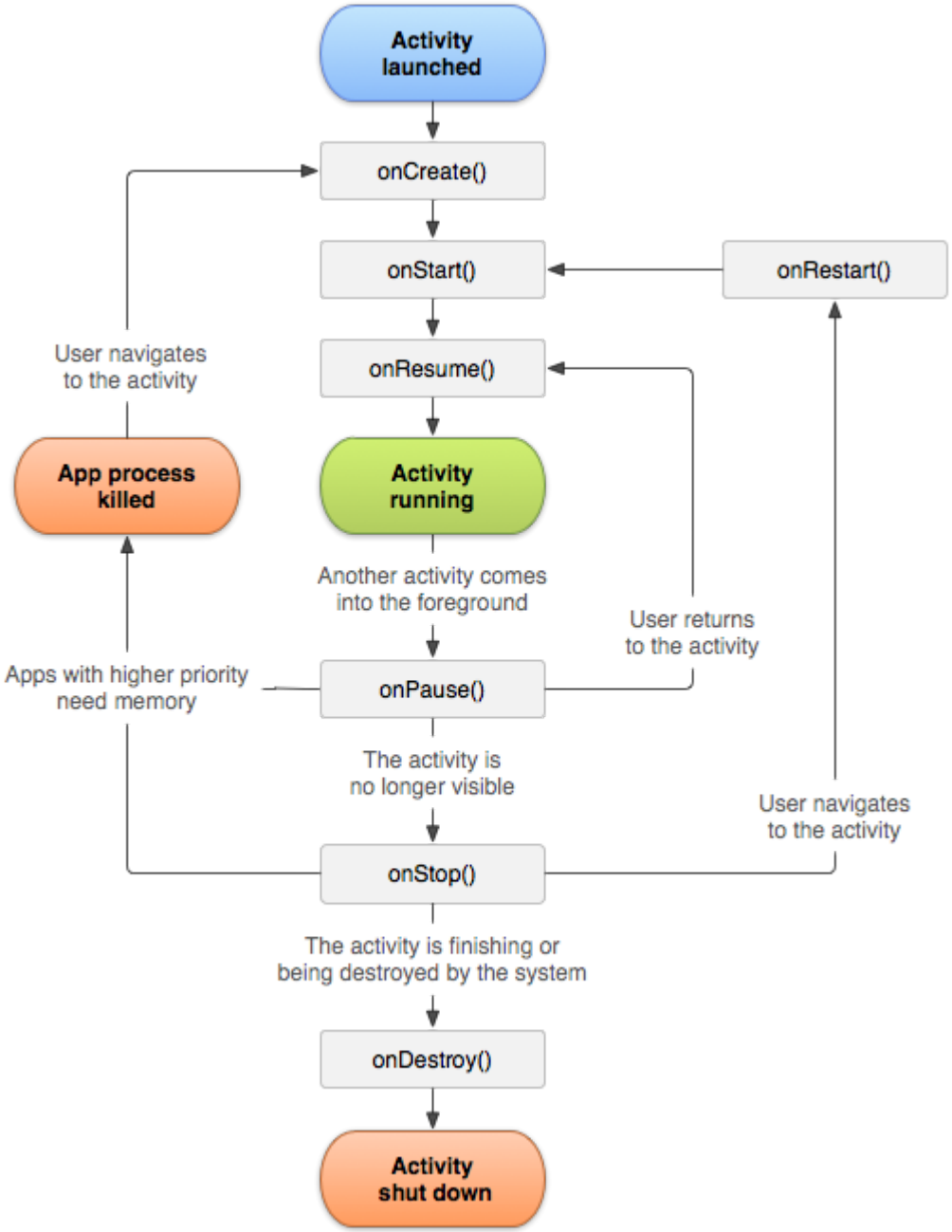
Activity

Una Activity en Android representa una única pantalla con una interfaz de usuario. En el contexto de la programación, una Activity pasa por varios estados reflejados en su ciclo de vida, manejados a través de métodos de callback como **onCreate()**, **onStart()**, **onResume()**, **onPause()**, **onStop()** y **onDestroy()**.

Cuando se retoma una Activity (por ejemplo, cuando el usuario regresa a ella después de haber utilizado otra aplicación), el sistema invoca el método onResume(). En este punto, la Activity entra en primer plano y el usuario puede interactuar con ella. Es aquí donde se debe reanudar cualquier operación suspendida que se haya pausado en onPause(), como reanudar la reproducción de animaciones o música, o la ejecución de hilos suspendidos. Las operaciones que consumen recursos intensivos que no son necesarias cuando la Activity no está en primer plano deben ser pausadas o detenidas en onPause(), y típicamente se reanudarían o reiniciarían en onResume().

Ciclo de vida de la activity

Una activity pasa por una serie de estados que son controlador por Android:



Las transiciones entre estados se procesan en el codigo mediante llamadas (callback) en el programa.

- **onCreate()** : Al crear la Activity. Es el momento de inicializar componentes
- **onStart()**: Al terminar onCreate se invoca onStart() y es visible para el usuario.
- **onResume()** Se invoca justo antes de que la app interacture con el usuaro. En este momento la Activity es la primera de la pila. En esta función se implementa la mayor parte de la lógica de la Activity.
- **onPause**:
- **onPause()** : El sistema llama onPause cuando la app pierde el foco o pasaal estado detenida.No debes usar onPause() para guardar datos de la aplicación o del usuario, realizar llamadas de red o ejecutar transacciones de base de datos. Para obtener más información sobre cómo guardar datos, consulta Cómo guardar y restablecer el estado de la actividad.
- **onStop()** : El sistema llama a onStop() cuando la actividad ya no es visible para el usuario, lo cual puede ocurrir porque se está eliminando la actividad, porque se inicia una nueva o porque una ya existente pasa al estado Reanudada y cubre la actividad que se detuvo. En todos estos casos, la actividad detenida ya no está visible en absoluto.

- **onRestart** : El sistema invoca esta devolución de llamada cuando una actividad en estado Detenida está por volver a iniciarse. onRestart() restaura el estado de la actividad desde el momento en que esta se detuvo.
- **onDestroy** : El sistema invoca esta devolución de llamada antes de que se elimine una actividad.

La llamada **onDestroy()** es la última que recibe la actividad. suele implementarse a fin de garantizar que todos los recursos de una actividad se liberen cuando esta, o el proceso que la contiene, se elimina.

Detalles y ejemplos en la documentación Android

Conservar el estado de una aplicación.

En Android, conservar el estado de una aplicación es crucial cuando se enfrentan cambios como rotaciones de pantalla, cambios de configuración o cuando el sistema destruye y recrea una Activity para reclamar recursos. Existen varias formas de conservar y restaurar el estado:

- **Guardar estado de instancia**
  - Utilizar `onSaveInstanceState(Bundle)` para guardar datos temporales.
  - Restaurar datos en `onCreate(Bundle)` o `onRestoreInstanceState(Bundle)`.
- **ViewModel**
  - Mantener datos a través de `ViewModel` para sobrevivir a cambios de configuración.
- **Persistencia de datos**
  - Almacenar datos más complejos o permanentes en bases de datos, `SharedPreferences` o en archivos.
- **Estado de vista guardado**
  - Guardar y restaurar el estado personalizado de vistas sobrescribiendo `onSaveInstanceState()` y `onRestoreInstanceState(Parcelable)`.
- **Jetpack DataStore**
  - Almacenar pares clave-valor o datos tipados de manera asíncrona y segura.
- **Estado de navegación**
  - Usar argumentos y estados guardados en la biblioteca de Navegación de Jetpack.
- **Procesos en segundo plano y servicios**
  - Utilizar servicios o WorkManager para ejecución de procesos largos y manejo de tareas en segundo plano.

Intent y filtros

En Android, un Intent es una descripción abstracta de una operación a realizar (se implementa en una clase). Actúa como un mensajero entre componentes de una aplicación (como actividades, servicios o receptores de difusión) o entre aplicaciones diferentes.

Usos de los Intent:

- Iniciar una actividad: Para iniciar una nueva Activity, se envía un Intent con el **contexto** y la **clase** de la Activity destino.
- Iniciar un **servicio**: Para iniciar o comunicarse con un Service, se utiliza un Intent para definir qué servicio se debe iniciar o vincular.
- Entregar un **broadcast**: Se utilizan para enviar un mensaje a través del sistema que cualquier BroadcastReceiver puede escuchar y responder.
- Comunicación de datos: Los Intent pueden llevar datos a través de "extras" (un Bundle de datos adicionales) donde se ponen pares clave-valor.
- Solicitar acciones: Pueden solicitar acciones de otros componentes, como tomar una foto, abrir una URL, o compartir contenido.

Los Intent también pueden incluir varias **banderas** (flags) que instruyen al sistema operativo sobre cómo lanzar una actividad y cómo relacionarla con el stack actual de actividades

Hay dos tipos de Intent en Android: explicitos y implicitos.

Intent explicitos

Se usan para iniciar un componente específico, **proporcionando la clase Java o Kotlin directamente**.
Se utilizan comúnmente para iniciar actividades dentro de la misma aplicación o para iniciar servicios que se ejecutan en segundo plano.

```
val intent = Intent(this, SomeOtherActivity::class.java)
startActivity(intent)
```

(Aunque sea ::class.java, sirve igual en kotlin)

Intent implicitos

No especifican directamente el componente; en lugar de eso, declaran una acción general para realizar, la cual permite que cualquier componente de cualquier aplicación que pueda manejar esa acción responda.

Se usan para realizar acciones con los componentes de otras aplicaciones, como abrir una URL en un navegador web o compartir datos.

```
val intent = Intent(Intent.ACTION_VIEW, Uri.parse("http://www.example.com"))
startActivity(intent)
```

O por ejemplo para usar la cámara de fotos

```
val takePictureIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
            startActivityForResult(takePictureIntent, REQUEST_IMAGE_CAPTURE)
```

Elementos

[TODO]

Propagación

[TODO]

Filtros

Los filtros en programación Android, conocidos como **Intent Filters**, son una funcionalidad que permite definir cómo se debe acceder a una actividad o servicio dentro de una aplicación. Un Intent Filter se define en el archivo de manifiesto de la aplicación (AndroidManifest.xml) y declara las capacidades de un componente, como por ejemplo, qué acciones puede realizar (como ver, editar, compartir) y qué tipos de datos puede manejar (como imágenes, texto, audio).

Cuando una aplicación emite un Intent (un mensaje asincrónico que permite solicitar una acción de otro componente de la aplicación o de otra aplicación), el sistema utiliza los filtros de intenciones para determinar qué componentes pueden responder a ese Intent. Esto incluye abrir una actividad, iniciar un servicio o entregar un mensaje a un receptor de emisión (BroadcastReceiver).

Los filtros de intención pueden especificar:

- **Acciones** (`<action>`): Las tareas generales que puede realizar un componente, como ACTION_VIEW, ACTION_EDIT.
- **Categorías** (`<category>`): Proporcionan información adicional sobre cómo se debe ejecutar una acción, como CATEGORY_LAUNCHER que indica que la actividad es la entrada principal de la aplicación.
- **Datos** (`<data>`): Especifican el tipo de datos que el componente puede manejar, incluyendo el tipo MIME y el esquema URI.

Ejemplo de definición de filtro de Intent

```
<activity android:name=".ExampleActivity" android:icon="@drawable/app_icon">
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="text/plain" />
    </intent-filter>
</activity>
```

Pending Intent