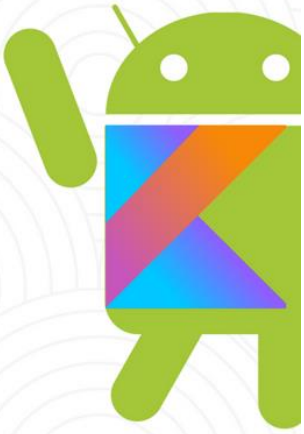


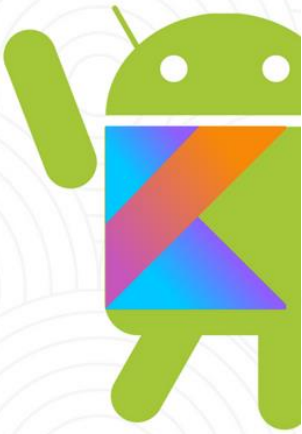
SENA

CENTRO DE COMERCIO Y TURISMO

INFORME ESTADOS ACTIVIDADES

FICHA

2469285



NOMBRE DE LA FORMACION

ADSI (ANÁLISIS Y DESARROLLO DE SISTEMAS DE INFORMACIÓN)

APRENDIZ

MAITE SALOME MARTINEZ RESTREPO

INSTRUCTOR

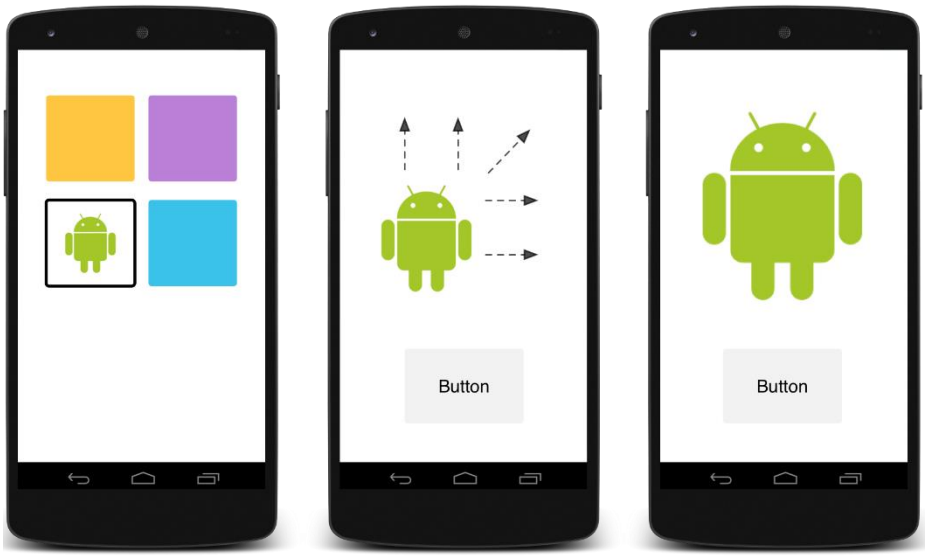
CRISTIAN DAVID HENAO HOYOS

ANDROID STUDIOS

Android Studio ofrece generadores de perfiles de rendimiento para que puedas realizar un seguimiento más fácil del uso de CPU y memoria de tu app, encontrar objetos desasignados, ubicar fugas de memoria, optimizar el rendimiento de los gráficos y analizar las solicitudes de red.

Uno de los conceptos principales en el desarrollo de aplicaciones móviles, es el concepto de Activity, siendo estas la base de la aplicación.

La clase Activity es un componente clave de una app para Android, y la forma en que se inician y se crean las actividades es una parte fundamental del modelo de aplicación de la plataforma. A diferencia de los paradigmas de programación en los que las apps se inician con un método main(), el sistema Android inicia el código en una instancia de Activity invocando métodos de devolución de llamada específicos que corresponden a etapas específicas de su ciclo de vida.

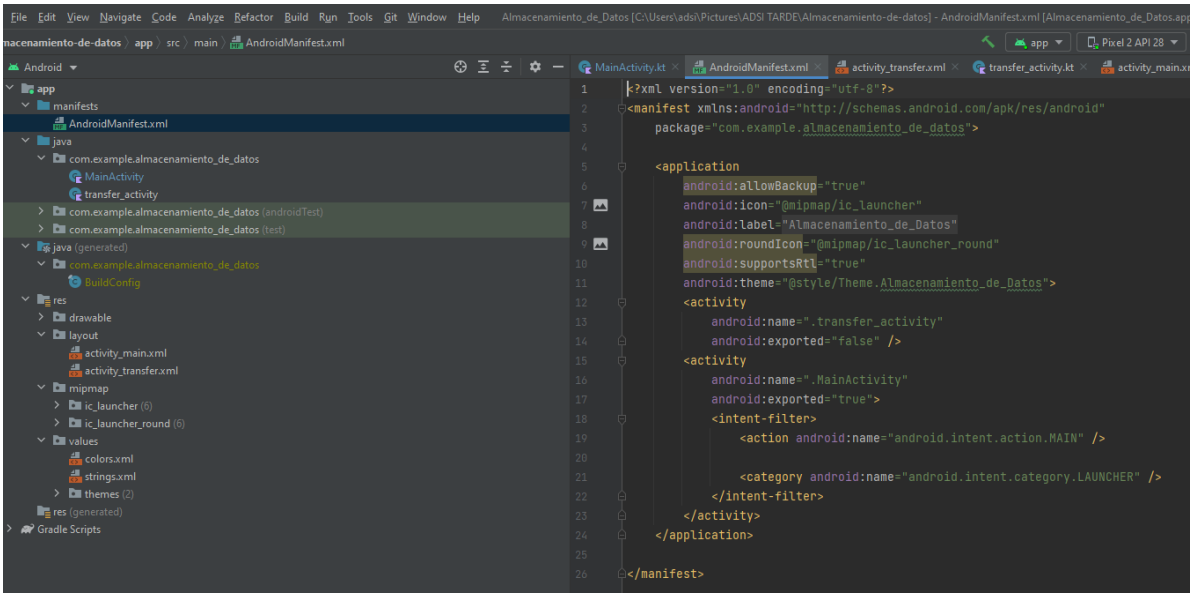


Está diseñada para facilitar este paradigma. Cuando una app invoca a otra, la app que realiza la llamada invoca una actividad en la otra, en lugar de a la app en sí. De esta manera, la actividad sirve como el punto de entrada para la interacción de una app con el usuario. Implementas una actividad como una subclase de la clase Activity.

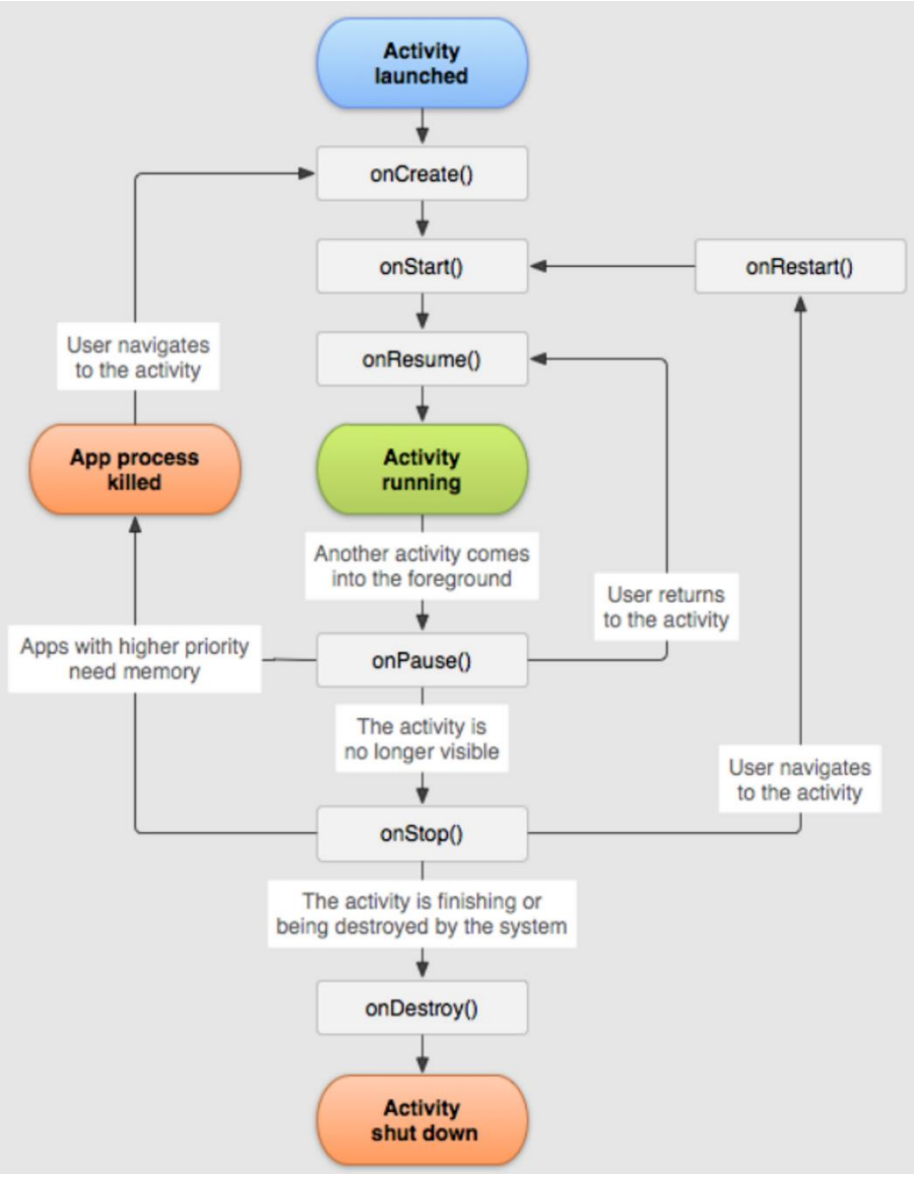
Una actividad proporciona la ventana en la que la app dibuja su IU. Por lo general, esta ventana llena la pantalla, pero puede ser más pequeña y flotar sobre otras ventanas. Generalmente, una actividad implementa una pantalla en una app. Por ejemplo, una actividad de una app puede implementar una pantalla Preferencias mientras otra implementa una pantalla Seleccionar foto.

En Android también se maneja un término conocido como Archivo Manifiesto (AndroidManifest.xml), es cual es un archivo eficaz en la plataforma Android que permite describir la funcionalidad y los requisitos de la aplicación en Android, se encuentra en la raíz de cualquier proyecto Android, es lo más parecido a una descripción completa de la aplicación (componentes, actividades, servicios, permisos, proveedores de contenido, permite parametrizar elementos claves de la aplicación, aquí se encontrará información como el id de la app, el nombre, el icono, las actividades que contiene, permisos entre otros elementos importantes.

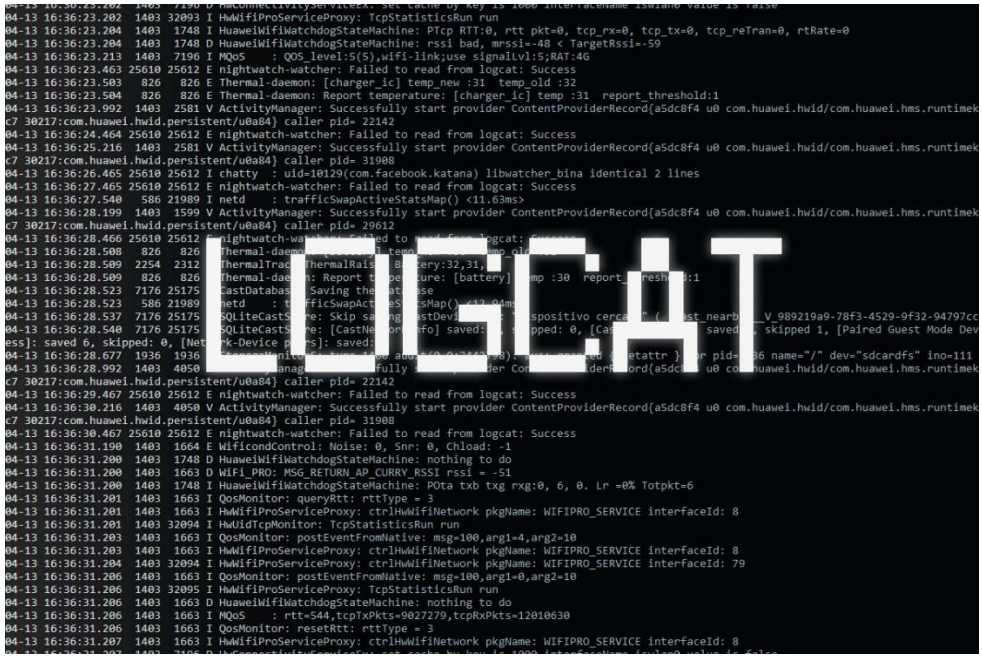
Cuando se crea una actividad, automáticamente androidStudio se encarga de parametrizarla en el archivo AndroidManifest.xml y allí se pueden agregar también filtros o propiedades para determinar acciones adicionales de la aplicación, por ejemplo en la imagen anterior se puede ver que el MainActivity es la actividad principal del sistema (la que primero se muestra) ya que se le agrega el <intent-filter> con la propiedad <action android:name="android.intent.action.MAIN" />



Existe también el ciclo de vida de las actividades en la cual hace alusión como su nombre lo dice, son los procesos por el cual pasa una Activity, en este caso la palabra correcta seria estados, es decir, pasa por varios estados durante su ejecución, estos estados se dan mediante la interacción del usuario con la Activity, por ejemplo cuando se ingresa, se sale, se ingresa nuevamente o cuando se cierra, Android nos permite controlar cada uno de estos estados mediante una serie de métodos definidos los cuales podemos implementar para gestionar el comportamiento, por ejemplo se puede pausar un reproductor de video cuando se cambia de pantalla o cancelar procesos de conexión cuando se sale de la aplicación.



Antes de ver el funcionamiento de los estados, veremos que es el logcat y cual es la utilidad a base de nuestra experiencia



El logcat es una herramienta de línea de comandos proporcionada por Android Studios que descarga un registro de los mensajes del sistema, incluidos los mensajes que se ha escrito desde la aplicación a través de los métodos Log proporcionados por la clase android.util.Log. Logcat es una ventana que muestra información detallada sobre lo que sucede en la aplicación como errores, advertencias, información de depuración, eventos del sistema y mensajes de registro que se haya creado en el código, algo bueno de esto es que podemos seleccionar diferentes niveles de registros, es decir, filtrar los mensajes de registro por etiquetas o prioridad, hasta podemos buscar mensajes específicos utilizando el cuadro de búsqueda como veremos mas adelante en los ejemplos, en conclusión, en eso se basa su utilizada en imprimir mensajes de depuración con diferentes niveles de importancia.

Ahora hablaremos acerca de cuales son los niveles de log en logcat y en que consiste cada uno de ellos

En Android existen cinco niveles de registro de mensajes que se utilizan con los métodos **Log.** Estos niveles se utilizan para indicar la importancia o la gravedad del mensaje que se está registrando. Los niveles de registro consisten en:

VERBOSE: Este nivel se utiliza para registrar mensajes detallados y de bajo nivel. Es el nivel más bajo de registro y se utiliza para mensajes que no son críticos, pero que pueden ser útiles para el desarrollo y la depuración. Para registrar un mensaje con este nivel, se utiliza el método Log.v().

DEBUG: Este nivel se utiliza para mensajes de depuración que son útiles para identificar problemas en la aplicación. Proporciona información detallada sobre el estado de la aplicación y el flujo de trabajo. Para registrar un mensaje con este nivel, se utiliza el método Log.d().

INFO: Este nivel se utiliza para mensajes informativos que indican el progreso de la aplicación o proporcionan información útil al usuario. Proporciona información importante pero no crítica. Para registrar un mensaje con este nivel, se utiliza el método Log.i().

WARN: Este nivel se utiliza para mensajes que indican una situación de advertencia en la aplicación. No son necesariamente errores, pero pueden indicar problemas potenciales que deben ser atendidos. Para registrar un mensaje con este nivel, se utiliza el método Log.w().

ERROR: Este nivel se utiliza para mensajes que indican una situación de error en la aplicación. Son mensajes críticos que indican un problema que necesita ser solucionado. Para registrar un mensaje con este nivel, se utiliza el método Log.e().

Entonces como vemos nos permiten identificar rápidamente los problemas y proporcionar información mas a fondo sobre lo que está sucediendo en la aplicación en tiempo de ejecución.

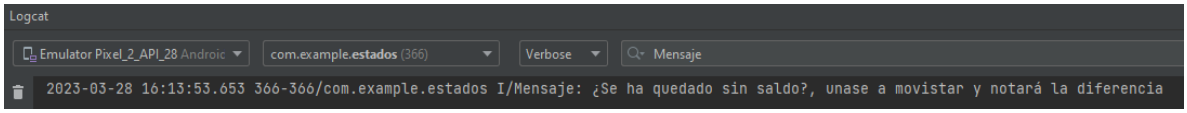
Veamos un resumen de cada uno de los estados en funcionamiento

1. **onCreate()**

   Este método es creado por defecto en la actividad y se activa cuando esta inicia con el fin de referenciar el layout correspondiente la parte gráfica mediante el método setContentView(), este es el primer método que se ejecuta.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    Log.i( tag: "Mensaje",  msg: "¿Se ha quedado sin saldo?, unase a movistar y notará la diferencia")
}
```

   Resultado del Logcat:

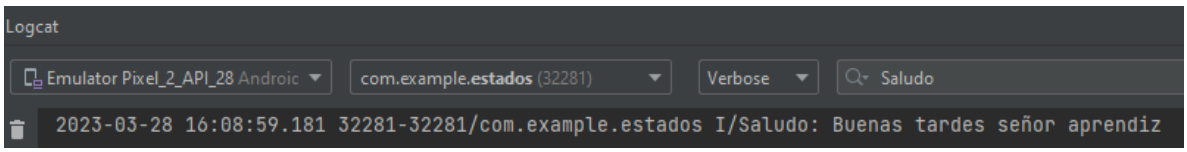


2. **onStart()**

   Al momento de cerrar el onCreate(), la actividad cambia su estado de creada a iniciada y se presenta al usuario, aquí es donde la vista y la actividad se hacen interactivas.

```
override fun onStart() {
    super.onStart()
    Log.i( tag: "Saludo",  msg: "Buenas tardes señor aprendiz")
}
```

   Resultado del Logcat:

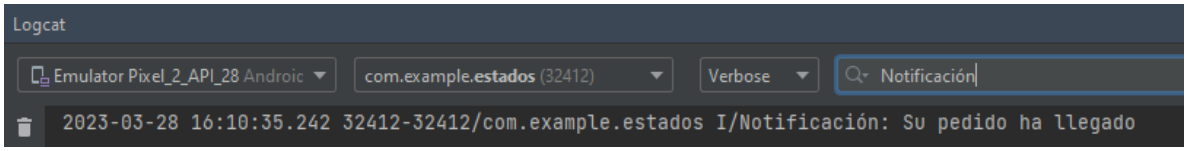


3. **onResume()**

Después de iniciada la actividad se cambia al estado onResume() qué se encarga de procesar la información de interacción con el usuario, aquí se reconoce y captura todo lo que el usuario ingresa.

```
override fun onResume() {
    super.onResume()
    Log.i( tag: "Notificación",  msg: "Su pedido ha llegado")
}
```

Resultado del Logcat:



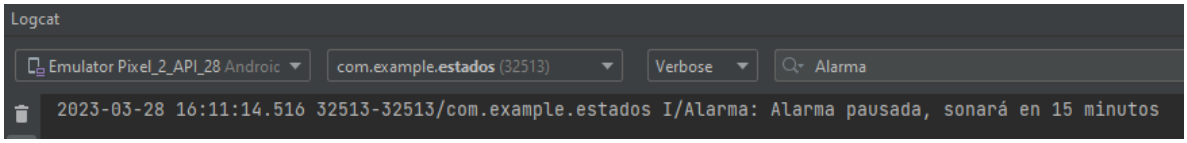
4. **onPause()**

En caso de qué la Activity pierda el foco y se encuentre detenida se llama al estado de Pausa, por ejemplo, cuando el usuario presione el botón de atrás o cambie de pantalla, significa que la actividad sigue parcialmente visible mientras se está saliendo de la actividad.

Desde este estado se puede regresar nuevamente a onResume() o a onStop()

```
override fun onPause() {
    super.onPause()
    Log.i( tag: "Alarma",  msg: "Alarma pausada, sonará en 15 minutos")
}
```

Resultado del Logcat:

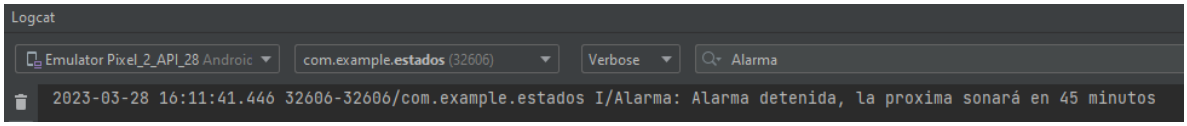


5. **onStop()**

   Este estado se inicia cuando la actividad ya no es visible para el usuario, puede darse porqué se elimina la actividad, se está reactivando una qué estaba detenida o porqué se está iniciando una nueva, en este caso la actividad en stop ya no es visible para el usuario, desde aquí se puede llamar a onRestart() o a onDestroy()

```
override fun onStop() {
    super.onStop()
    Log.i( tag: "Alarma",  msg: "Alarma detenida, la proxima sonará en 45 minutos")
}
```

   Resultado del Logcat:

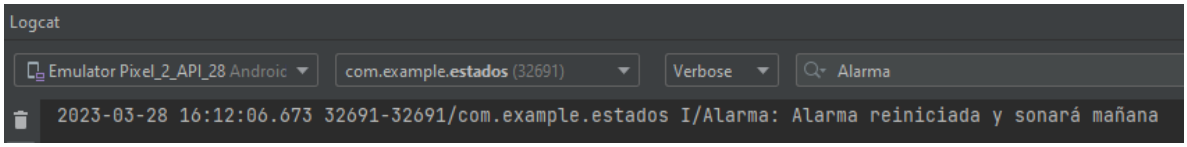


6. **onRestart()**

   El sistema ingresa a este estado cuando una actividad detenida va a iniciarse nuevamente, aquí se restaura el estado de la actividad desde el momento en que se detuvo y se hace el llamado a onStart()

```
override fun onRestart() {
    super.onRestart()
    Log.i( tag: "Alarma",  msg: "Alarma reiniciada y sonará mañana")
}
```

   Resultado del Logcat:



7. **onDestroy()**

   Finalmente, el sistema invoca a este método antes de eliminar la actividad, este es el último estado por el que pasa la actividad y se implementa cuando

se deba garantizar el cierre o liberación de todos los recursos de una actividad

```
override fun onDestroy() {
    super.onDestroy()
    Log.i( tag: "Alarma",  msg: "Alarma apagada")
}
```

Resultado del Logcat: