	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN	
CICLO: I/2021	GUIA DE LABORATORIO #03	
	Nombre de la Práctica:	Ciclo de vida de una app
	MATERIA:	Desarrollo de Software para Móviles

I. OBJETIVOS

1. Conocer el concepto de Activity o Actividad.
2. Conocer el ciclo de vida de una aplicación Android.
3. Crear una aplicación Android e identificar en qué momento se ejecutan las diferentes etapas del ciclo de vida utilizando Lifecycle-Aware components.

II. INTRODUCCION TEORICA

Al hablar del ciclo de vida de una aplicación Android estamos hablando en realidad del ciclo de vida de una Activity, por lo que primeramente es necesario responder la pregunta: ¿Qué es una **Activity**?

Una **Activity o Actividad** en una aplicación Android es una pantalla. Es el componente de una aplicación donde están presentes elementos de interfaz gráfica (botones, listados, cajas de texto, etc) y donde el usuario puede interactuar con la aplicación haciendo cosas como: tomar una foto, enviar un correo, etc. Imagina **tu aplicación como la ventana de un navegador** y la **Activity cómo la pestaña donde carga cualquier sitio web**.

Un ciclo de vida es una serie de estados secuenciales por los que una Activity pasa desde que es creada hasta que es destruida y en los que en cada uno se ejecutan ciertas acciones necesarias para el correcto funcionamiento de la aplicación.

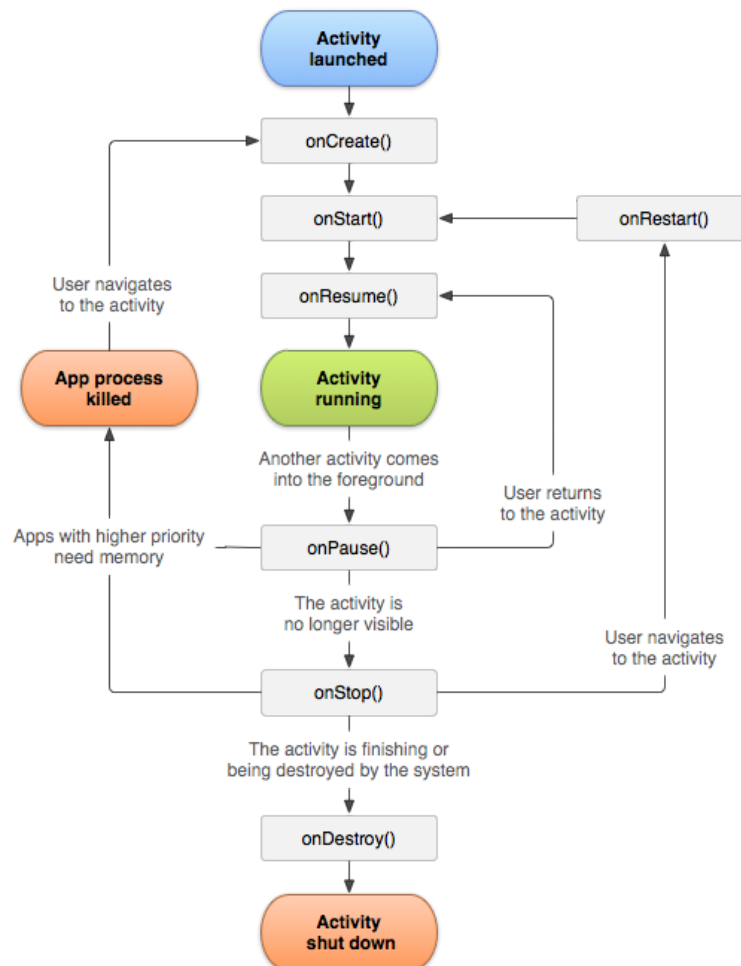
Cuando un usuario navega por tu app, sale de ella y vuelve a entrar, las instancias de tus Activity pasan por diferentes estados de su ciclo de vida. La clase Activity posee una serie de **callbacks** (estos funcionan como triggers o disparadores) que permiten a la actividad saber si cambio de estado y en qué estado se encuentra, así que es posible conocer si la actividad se está creando, reanudando o destruyendo.

Dentro de los métodos callback del ciclo de vida, puedes declarar el comportamiento que tendrá tu actividad cuando el usuario la abandone y la reanude. Por ejemplo, si creas un reproductor de vídeo en streaming, puedes pausar el video y cancelar la conexión de red cuando el usuario cambia a otra app. Cuando el usuario vuelve, puedes volver a establecer

la conexión con la red y permitir que el usuario reanude el video desde el mismo punto. En otras palabras, cada callback te permite realizar un trabajo específico que es apropiado para un cambio de estado en particular.

Para navegar por las transiciones entre las etapas del ciclo de vida de una actividad, la clase Activity proporciona un conjunto básico de seis métodos: **onCreate()**, **onStart()**, **onResume()**, **onPause()**, **onStop()** y **onDestroy()**. El sistema invoca cada una de estos métodos cuando una operación entra en un nuevo estado.

A continuación se muestra una representación visual de este paradigma.



onCreate

Debes implementar este método, que se activa cuando el sistema crea la actividad por primera vez. Cuando se crea la actividad, esta entra en el estado **Created**. En el método `onCreate()`, ejecutas la lógica de arranque básica de la aplicación **que debe ocurrir una sola vez en toda la vida de la actividad**. Generalmente este método es utilizado para inicializar la interfaz gráfica de tu aplicación y/o preparar los datos que utilizas en la misma.

onStart

La llamada onStart() hace que el usuario pueda ver la actividad mientras la app se prepara para que esta entre en primer plano y se convierta en interactiva.

El método onStart() se completa muy rápido y, al igual que con el estado Created, la actividad no permanece en el estado **Started**. Una vez finalizada esta devolución de llamada, la actividad entra en el estado **Resumed**, y el sistema invoca el método onResume().

onResume

Cuando la actividad entra en el estado **Resumed**, pasa al primer plano y, a continuación, el sistema invoca la devolución de llamada onResume(). Este es el estado en el que la app interactúa con el usuario. La app permanece en este estado hasta que ocurre algún evento que la quita de foco. Tal evento podría ser, por ejemplo, recibir una llamada telefónica, que el usuario navegue a otra actividad o que se apague la pantalla del dispositivo.

Aquí es donde los componentes del ciclo de vida pueden habilitar cualquier funcionalidad que necesite ejecutarse mientras el componente esté visible y en primer plano, como, por ejemplo, iniciar una vista previa de la cámara.

Cuando se produce un evento de interrupción, la actividad entra en el estado **Paused** y el sistema invoca la devolución de llamada onPause().

Si la actividad regresa al estado **Resumed** desde **Paused**, el sistema volverá a llamar al método onResume().

onPause

El sistema llama a este método a modo de primera indicación de que el usuario está abandonando tu actividad (aunque no siempre significa que está finalizando la actividad); esto indica que la actividad ya no está en primer plano (aunque puede seguir siendo visible si el usuario está en el modo multiventana). Utiliza el método onPause() para pausar o ajustar las operaciones que no deben continuar (o que deben continuar con moderación) mientras Activity se encuentra en estado **Paused** y que esperas reanudar en breve.

También puedes utilizar el método onPause() para liberar recursos del sistema, controladores de sensores (como el GPS) o cualquier otro recurso que pueda afectar la duración de la batería mientras tu actividad esté en pausa y el usuario no los necesite. Sin embargo, como se mencionó antes en la sección onResume(), una actividad con el estado Paused puede ser completamente visible si está en el modo multiventana. Por eso, deberías considerar usar onStop() en lugar de onPause() para liberar o ajustar por completo los recursos y operaciones relacionados con la IU a fin de admitir mejor el modo multiventana.

La ejecución de `onPause()` es muy breve y no necesariamente permite disponer de tiempo suficiente para realizar operaciones seguras. Por esta razón, **no debes utilizar `onPause()` para guardar los datos de la aplicación o del usuario, realizar llamadas de red o ejecutar transacciones de la base de datos, ya que es posible que no se complete dicho trabajo antes de que finalice el método.**

En su lugar, debes realizar operaciones de finalización de cargas pesadas durante `onStop()`.

onStop

Cuando el usuario ya no puede ver tu actividad, significa que ha entrado en el estado **Stopped**, y el sistema invoca la devolución de llamada `onStop()`. Esto puede ocurrir, por ejemplo, cuando una actividad recién lanzada cubre toda la pantalla. El sistema también puede llamar a `onStop()` cuando haya terminado la actividad y esté a punto de finalizar.

En el método `onStop()`, la app debe liberar o ajustar los recursos que no son necesarios mientras no sea visible para el usuario. Por ejemplo, tu app podría pausar animaciones o cambiar de actualizaciones de ubicación detalladas a más generales.

También debes utilizar `onStop()` para realizar operaciones de finalización con un uso relativamente intensivo de la CPU. Por ejemplo, si no encuentras un momento más oportuno para guardar información en una base de datos, puedes hacerlo en `onStop()`.

Desde el estado **Stopped**, la actividad regresa a interactuar con el usuario o se termina de ejecutar y desaparece. Si la actividad regresa, el sistema invoca a `onRestart()`. Si se terminó de ejecutar Activity, el sistema llamará a `onDestroy()`.

onDestroy

Se llama a `onDestroy()` antes de que finalice la actividad. El sistema invoca esta devolución de llamada por los siguientes motivos:

1. La actividad está terminando (debido a que el usuario la descarta por completo o a que se llama a `finish()`).
2. El sistema está finalizando temporalmente la actividad debido a un cambio de configuración (como la rotación del dispositivo o el modo multiventana).

Si la actividad está terminando, `onDestroy()` es la devolución de llamada del ciclo de vida final que recibe la actividad. Si se llama a `onDestroy()` como resultado de un cambio de configuración, el sistema crea inmediatamente una nueva instancia de actividad y luego llama a `onCreate()` en esa nueva instancia en la nueva configuración.

La devolución de llamada `onDestroy()` debe liberar todos los recursos que aún no han sido liberados por devoluciones de llamada anteriores, como `onStop()`.

Lifecycle-Aware Components

Los componentes optimizados para ciclos de vida (**Lifecycle-Aware Components**) realizan acciones como respuesta a un cambio en el estado del ciclo de vida de otro componente, como actividades o fragmentos. Estos componentes te ayudan a crear un código mejor organizado, y a menudo más liviano, que resulta más fácil de mantener.

Un patrón común consiste en implementar las acciones de los componentes dependientes en los métodos del ciclo de vida de actividades y fragmentos. Sin embargo, este patrón genera una organización deficiente del código y la proliferación de errores. Al usar componentes optimizados para ciclos de vida, puedes sacar el código de los componentes dependientes que se encuentra en los métodos del ciclo de vida y colocarlo en los propios componentes.

El paquete **androidx.lifecycle** incluye interfaces y clases que te permiten compilar componentes optimizados para ciclos de vida; es decir, componentes que pueden ajustar automáticamente su comportamiento en función del estado actual del ciclo de vida de una actividad o un fragmento.

Una clase puede supervisar el estado del ciclo de vida del componente agregando anotaciones a sus métodos. Luego, puedes agregar un observador. Para ello, llama al método **addObserver()** de la clase **Lifecycle** y pasa una instancia de tu observador, como se muestra en el siguiente ejemplo:

```
public class MyObserver implements LifecycleObserver {
    @OnLifecycleEvent(Lifecycle.Event.ON_RESUME)
    public void connectListener() {
        ...
    }

    @OnLifecycleEvent(Lifecycle.Event.ON_PAUSE)
    public void disconnectListener() {
        ...
    }
}

getLifecycle().addObserver(new MyObserver());
```

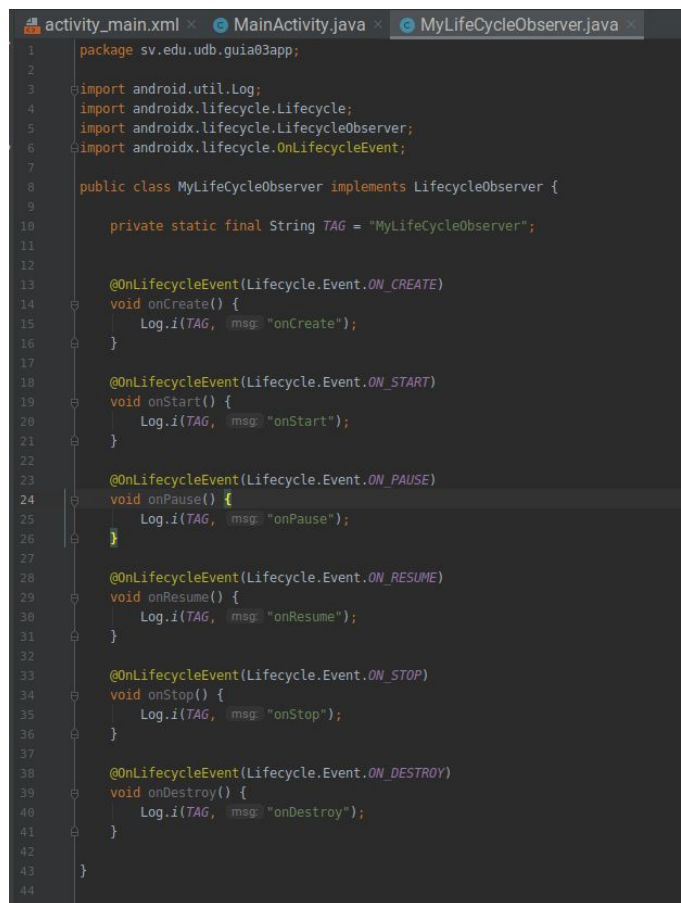
III. PROCEDIMIENTO

Para realizar la práctica debe haber completado con éxito la guía anterior **Introducción a Android**.

1. Ejecuta Android Studio y crea un nuevo proyecto. En **Project Template** selecciona **Empty Activity** y presiona **Next**.
2. Como **Name** coloca **Guia03App**, en **Package name** escribe **sv.edu.udb.guia03app** en **Save location** escoge la carpeta de tu preferencia, en Language elige **Java** y el **Minimum SDK** selecciona **API 16: Android 4.1 (Jelly Bean)**

Nota: En esta práctica se hará uso del lenguaje de programación Java pero, se invita al estudiante a realizarla también utilizando Kotlin.

3. Presiona **Finish**.
4. Haz clic derecho sobre tu package y selecciona **New** luego **Java class** y como nombre coloca **MyLifecycleObserver**.
5. Escribe el siguiente código en la clase recién creada.

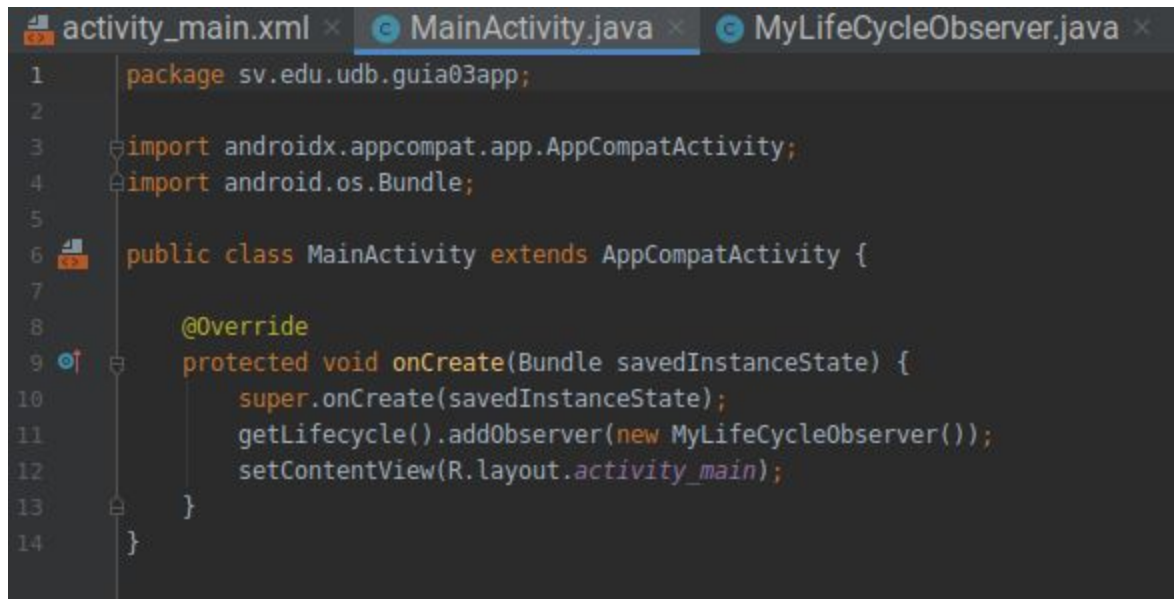


```

1 package sv.edu.udb.guia03app;
2
3 import android.util.Log;
4 import androidx.lifecycle.Lifecycle;
5 import androidx.lifecycle.LifecycleObserver;
6 import androidx.lifecycle.OnLifecycleEvent;
7
8 public class MyLifecycleObserver implements LifecycleObserver {
9
10     private static final String TAG = "MyLifecycleObserver";
11
12     @OnLifecycleEvent(Lifecycle.Event.ON_CREATE)
13     void onCreate() {
14         Log.i(TAG, "onCreate");
15     }
16
17     @OnLifecycleEvent(Lifecycle.Event.ON_START)
18     void onStart() {
19         Log.i(TAG, "onStart");
20     }
21
22     @OnLifecycleEvent(Lifecycle.Event.ON_PAUSE)
23     void onPause() {
24         Log.i(TAG, "onPause");
25     }
26
27     @OnLifecycleEvent(Lifecycle.Event.ON_RESUME)
28     void onResume() {
29         Log.i(TAG, "onResume");
30     }
31
32     @OnLifecycleEvent(Lifecycle.Event.ON_STOP)
33     void onStop() {
34         Log.i(TAG, "onStop");
35     }
36
37     @OnLifecycleEvent(Lifecycle.Event.ON_DESTROY)
38     void onDestroy() {
39         Log.i(TAG, "onDestroy");
40     }
41
42 }
43
44

```

6. Abre la clase **MainActivity** y editala agregando las siguientes líneas de código:

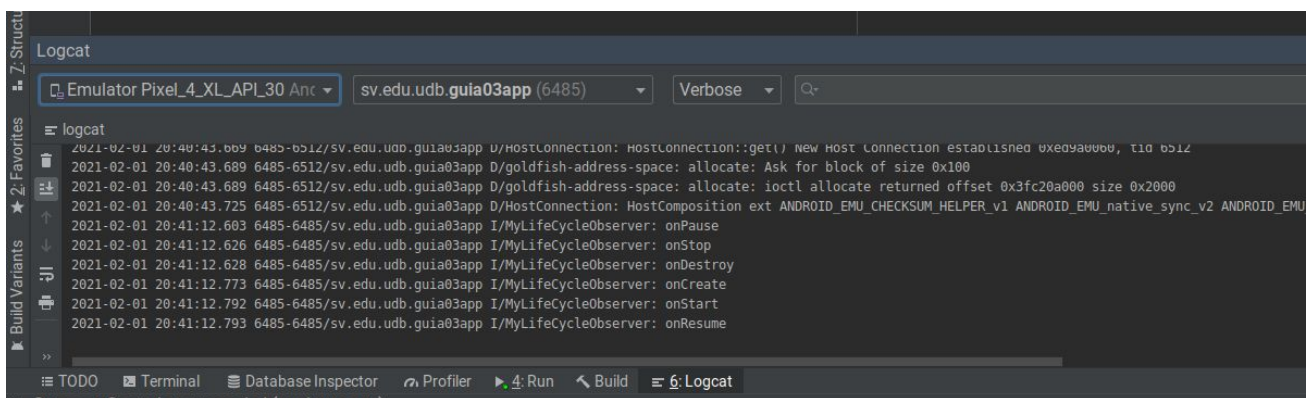


```

1  package sv.edu.udb.guia03app;
2
3  import androidx.appcompat.app.AppCompatActivity;
4  import android.os.Bundle;
5
6  public class MainActivity extends AppCompatActivity {
7
8      @Override
9      protected void onCreate(Bundle savedInstanceState) {
10         super.onCreate(savedInstanceState);
11         getLifecycle().addObserver(new MyLifecycleObserver());
12         setContentView(R.layout.activity_main);
13     }
14 }
    
```

7. Presiona **Run**.

En la parte inferior del Android Studio en la sección **Logcat** podrás ver una serie de mensajes que indican el estado actual del ciclo de vida en el que la aplicación se encuentra. Si giras el dispositivo, minimizas la aplicación o la cierras y vuelves a abrir podrás apreciar el ciclo de vida de la actividad por completo.



IV. DISCUSIÓN DE RESULTADOS

1. Crea una aplicación Android que cumpla con los siguientes requisitos:

- La interfaz gráfica será una única pantalla (una actividad) que contará con un **TextView** al centro con el siguiente texto inicial “0” y un **Button** debajo de este con el texto “+1”.
- Al presionar el botón el mensaje del TextView se modificara aumentando su valor en 1. Siendo su valor máximo 9 al superar ese valor el mensaje se reiniciará y volverá a mostrar “0”.
- Cuando la aplicación gire o pase a segundo plano, al recuperar el focus el TextView deberá seguir mostrando el número que poseía antes de girar. Es decir que la aplicación debe programarse para preservar el estado de la UI.

Para la realización de esta discusión de resultados se recomienda el uso de **ViewModel** y **LiveData**.

V. BIBLIOGRAFÍA

- Documentación Oficial Android
<https://developer.android.com/guide/components/activities/activity-lifecycle>
- ViewModel
<https://developer.android.com/topic/libraries/architecture/viewmodel>