


| | | | |
|---|--|---|---|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | GUIA DE LABORATORIO N° 2 |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |

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.
4. Conocer los tipos de componentes en Android
5. Crear una aplicación en Android, donde se comuniquen dos Actividades.
6. Conocer el archivo manifiesto, para los permisos en Android

II. INTRODUCCIÓN TEÓRICA – PRIMERA PARTE

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 o devolución de llamada te permite realizar un trabajo específico que es apropiado para un cambio de estado en particular. Hacer el trabajo preciso en el momento adecuado y administrar las transiciones correctamente hace que tu app sea más sólida y eficiente.

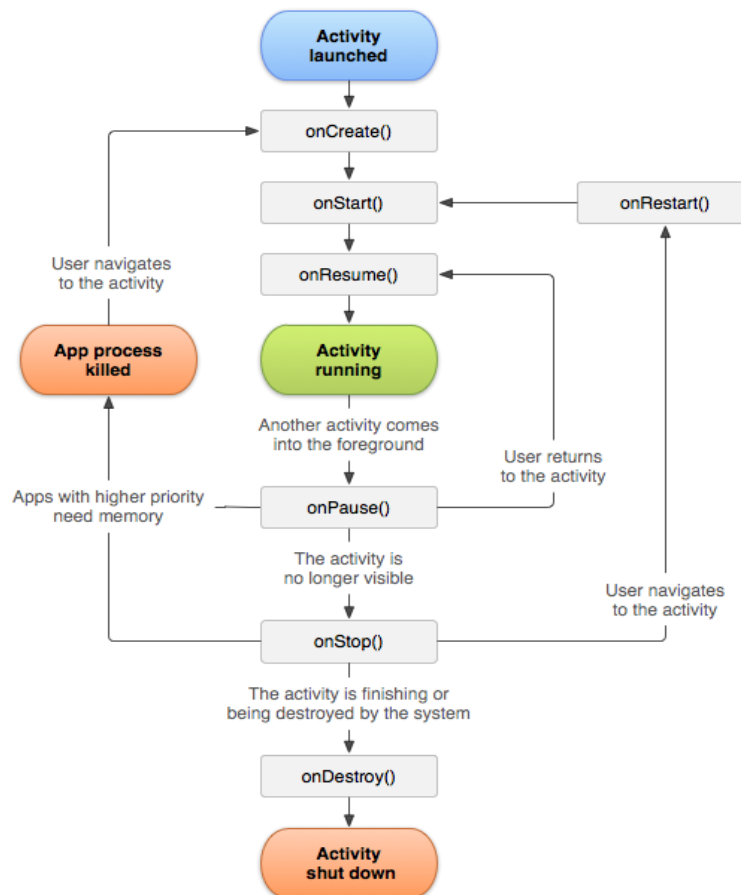
Por ejemplo, una buena implementación de las devoluciones de llamada de un ciclo de vida puede ayudar a garantizar que tu app:

- No falle si el usuario recibe una llamada telefónica o cambia a otra app mientras usa la tuya.
- No consuma recursos valiosos del sistema cuando el usuario no la use de forma activa.
- No pierda el progreso del usuario si este abandona tu app y regresa a ella posteriormente.
- No falle ni pierda el progreso del usuario cuando se gire la pantalla entre la orientación horizontal y la vertical.

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

| | | | |
|---|--|---|---|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | GUIA DE LABORATORIO N° 2 |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |

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.

Es el lugar donde se incluye la mayor de la inicialización. Tareas prioritarias como:

- Poner atributos de diseño a la UI de la actividad con setContentView().
- Agregar fragmentos.
- Obtener referencias de Views con findViewById().
- Enlazar datos a los View.
- Iniciar consultas iniciales a fuentes de datos.
- Crear instancias de los componentes de tu arquitectura.

➤ onStart()

La llamada onStart() hace que el usuario pueda ver la actividad mientras la app se prepara para que esta entre

| | | | |
|---|--|---|---|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | GUIA DE LABORATORIO N° 2 |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |

en primer plano y se convierta en interactiva. Por ejemplo, este método es donde la app inicializa el código que mantiene la IU (Interfaz de Usuario).

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.

Cuando la actividad pase al estado de pausa, cualquier componente que priorice el ciclo de vida vinculado al ciclo de vida de la actividad recibirá el evento ON_PAUSE. Aquí es donde los componentes del ciclo de vida pueden detener cualquier funcionalidad que no necesite ejecutarse mientras el componente no esté en primer plano, como detener una vista previa de la cámara.

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**

| | | | |
|--|--|---|---|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | GUIA DE LABORATORIO N° 2 |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |

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().

La finalización del método onPause() no significa que la actividad abandone el estado Paused. Más bien, la actividad permanecerá en ese estado hasta que se reanude o se vuelva completamente invisible para el usuario. Si se reanuda la actividad, el sistema volverá a invocar la devolución de llamada onResume(). Si la actividad regresa del estado Paused a Resumed, el sistema mantendrá la instancia Activity en la memoria y la volverá a llamar cuando invoque onResume(). En esta situación, no es necesario que reinicialices los componentes que se crearon durante los métodos de devolución de llamada que llevan al estado Resumed. Si la actividad se vuelve completamente invisible, el sistema llamará a 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.

Cuando la actividad pase al estado Stopped, cualquier componente que priorice el ciclo de vida vinculado al de la actividad recibirá el evento ON_STOP. Aquí es donde los componentes del ciclo de vida pueden detener cualquier funcionalidad que no necesite ejecutarse mientras el componente no sea visible en la pantalla.

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. Usar onStop() en lugar de onPause() garantiza que continúe el trabajo relacionado con la IU, incluso cuando el usuario esté viendo tu actividad en el modo multiventana.


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 onStart(). 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).

| | | | |
|--|--|---|---|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | GUIA DE LABORATORIO N° 2 |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |

Cuando la actividad pase al estado Destroyed, cualquier componente que priorice el ciclo de vida vinculado al de la actividad recibirá el evento ON_DESTROY. Aquí es donde los componentes del ciclo de vida pueden recuperar cualquier elemento que se necesite antes de que finalice el objeto Activity.

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.

Lifecycle

Lifecycle es una clase que mantiene la información sobre el estado del ciclo de vida de un componente (como una actividad o un fragmento) y permite que otros objetos observen este estado. Lifecycle usa dos enumeraciones principales para realizar un seguimiento del estado del ciclo de vida de su componente asociado:


➤ Evento

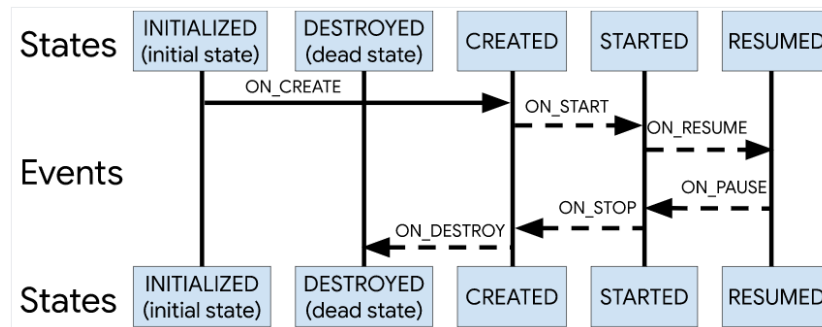
Los eventos de ciclo de vida que se despachan desde el framework y la clase Lifecycle. Estos eventos se asignan a eventos de devolución de llamada en actividades y fragmentos.

➤ Estado

Es el estado actual del componente al que le hace un seguimiento el objeto Lifecycle.

A continuación se muestra un diagrama sobre los estados y eventos que componen el ciclo de vida de una actividad en Android.

| | | | |
|---|--|---|---|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |
| | | | GUIA DE LABORATORIO N° 2 |



Una clase puede supervisar el estado del ciclo de vida del componente agregando anotaciones a sus métodos. Luego, puedes agregar un observador.

La interfaz LifecycleObserver representa al observador encargado de monitorear el estado del ciclo de vida de un componente a través de la anotación @OnLifecycleEvent.

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() {
        ...
    }
}

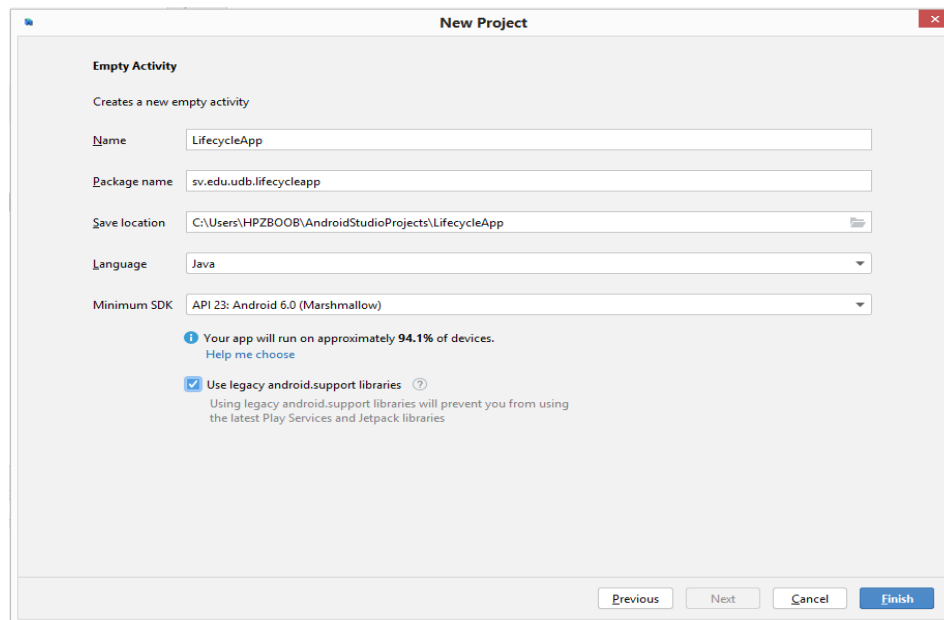
getLifecycle().addObserver(new MyObserver());

```

III. DESARROLLO DE PRÁCTICA – PRIMERA PARTE

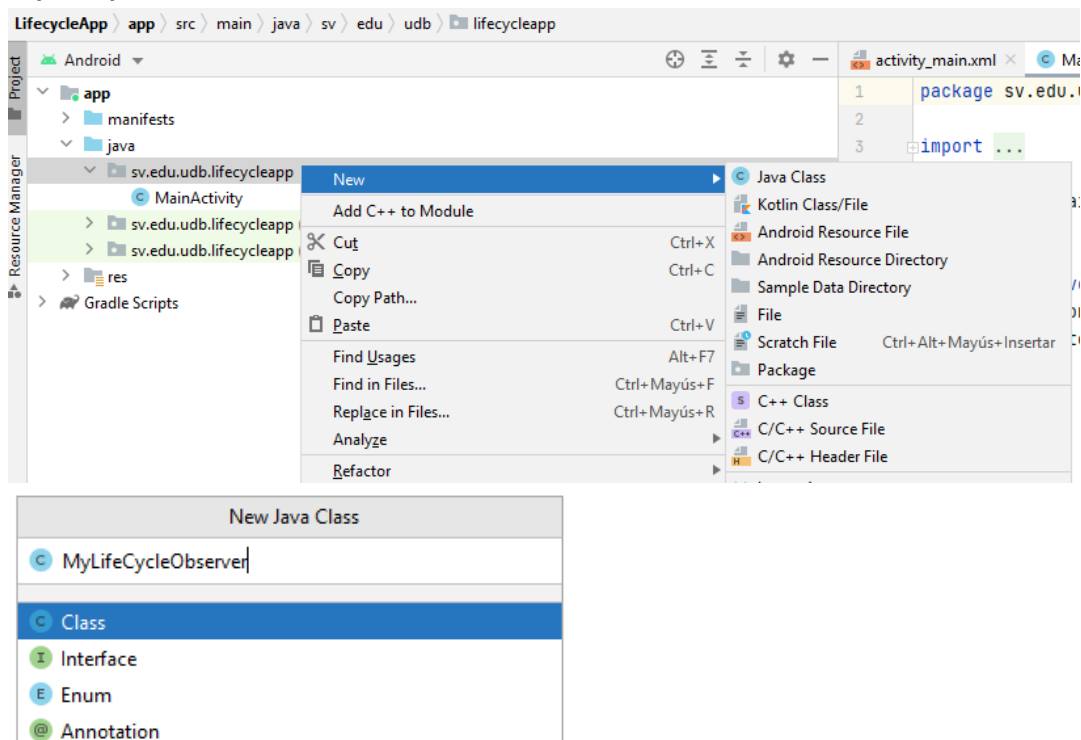
1. Ejecuta Android Studio y crea un nuevo proyecto. En **Project Template** selecciona **Empty Activity** y presiona **Next**.
2. Como **Name** coloca **LifecycleApp**, en **Package name** escribe **sv.edu.udb.lifecycleapp** en **Save location** escoge la carpeta de tu preferencia, en Language elige **Java** y el **Minimum SDK** selecciona **API 23: Android 6.0 (Marshmallow)**
3. Presiona **Finish**


| | | | |
|---|--|---|---|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | GUIA DE LABORATORIO N° 2 |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |



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

- Haz clic derecho sobre tu package y selecciona **New** luego **Java class** y como nombre coloca **MyLifecycleObserver**.




| | | | |
|---|--|---|---|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | GUIA DE LABORATORIO N° 2 |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |

5. Escribe el siguiente código en la clase recién creada.

```

1  package sv.edu.udb.lifecycleapp;
2
3  import android.arch.lifecycle.Lifecycle;
4  import android.arch.lifecycle.LifecycleObserver;
5  import android.arch.lifecycle.OnLifecycleEvent;
6  import android.util.Log;
7  import android.widget.Toast;
8
9  public class MyLifecycleObserver implements LifecycleObserver {
10
11      private static final String TAG = "MyLifecycleObserver";
12      private MainActivity obj;
13
14      public MyLifecycleObserver(MainActivity mainActivity) {
15          this.obj = mainActivity;
16      }
17
18      @OnLifecycleEvent(Lifecycle.Event.ON_CREATE) void onCreate(){
19          Log.i(TAG, msg: "OnCreate");
20          Toast.makeText(this.obj, text: "OnCreate", Toast.LENGTH_SHORT).show();
21      }
22
23      @OnLifecycleEvent(Lifecycle.Event.ON_START) void onStart(){
24          Log.i(TAG, msg: "OnStart");
25          Toast.makeText(this.obj, text: "OnStart", Toast.LENGTH_SHORT).show();
26      }
27
28      @OnLifecycleEvent(Lifecycle.Event.ON_RESUME) void onResume(){
29          Log.i(TAG, msg: "OnResume");
30          Toast.makeText(this.obj, text: "OnResume", Toast.LENGTH_SHORT).show();
31      }
32
33      @OnLifecycleEvent(Lifecycle.Event.ON_STOP) void onStop(){
34          Log.i(TAG, msg: "OnStop");
35          Toast.makeText(this.obj, text: "OnStop", Toast.LENGTH_SHORT).show();
36      }
37
38      @OnLifecycleEvent(Lifecycle.Event.ON_DESTROY) void onDestroy(){
39          Log.i(TAG, msg: "OnDestroy");
40          Toast.makeText(this.obj, text: "OnDestroy", Toast.LENGTH_SHORT).show();
41      }
42  }

```


| | | | |
|---|--|---|---|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | GUIA DE LABORATORIO N° 2 |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |

Nota: Se utilizara el objeto Toast para desplegar mensajes emergentes en nuestra interfaz, en conjunto utilizaremos la clase Log para crear mensajes de registro que se muestran en logcat.

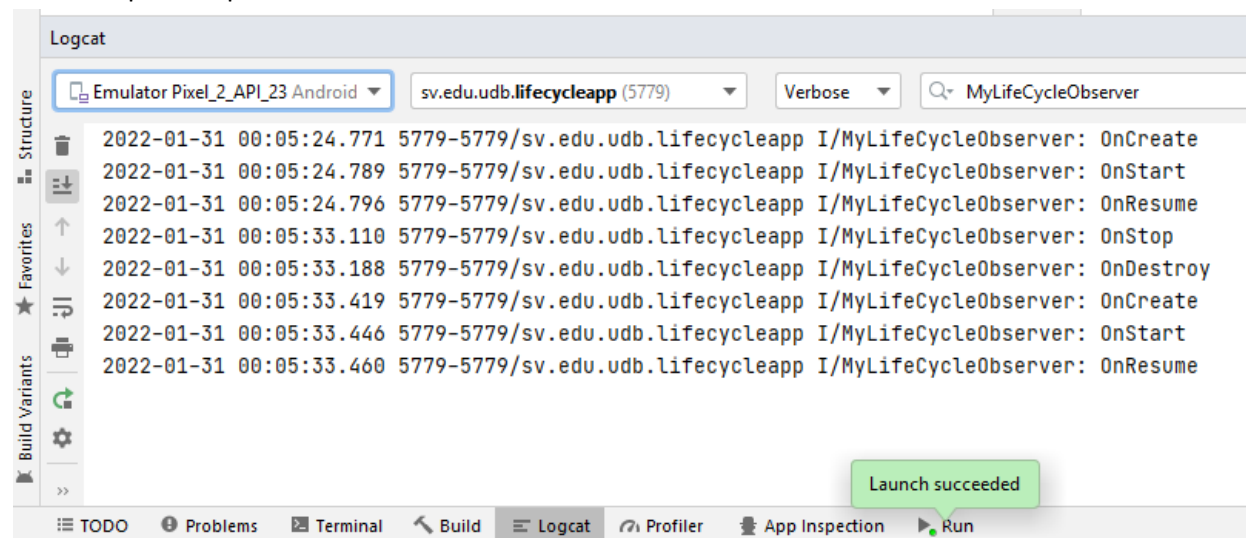
6. Abre la clase **MainActivity** y editala agregando las siguientes líneas de código:


```

1  package sv.edu.udb.lifecycleapp;
2
3  import android.support.v7.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( MainActivity: this));
12         setContentView(R.layout.activity_main);
13     }
14 }

```

7. Ejecute su aplicación, 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.



| | | | |
|---|--|---|---|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | GUIA DE LABORATORIO N° 2 |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |

IV. INTRODUCCIÓN TEÓRICA – SEGUNDA PARTE

Componentes de la aplicación

Los componentes de la aplicación son bloques de creación esenciales de una aplicación para Android. Cada componente es un punto de entrada por el que el sistema o un usuario ingresan a tu aplicación. Algunos componentes dependen de otros.

Las aplicaciones tienen cuatro tipos de componentes diferentes:

- **Actividades (Activities)**
- **Servicios (Services)**
- **Receptores de emisiones (Emission receivers -> BroadcastReceiver)**
- **Proveedores de contenido (Content providers)**

Cada tipo tiene un fin específico y un ciclo de vida característico que define cómo se crea y se destruye el componente.

➤ **Actividades**


Una actividad es el punto de entrada de interacción con el usuario. Representa una pantalla individual con una interfaz de usuario. Por ejemplo, una aplicación de correo electrónico tiene una actividad que muestra una lista de los correos electrónicos nuevos, otra actividad para redactar el correo electrónico y otra actividad para leerlo.

Si bien las actividades funcionan juntas para ofrecer una experiencia de usuario uniforme en la aplicación de correo electrónico, cada una es independiente de las demás. De esta manera, una aplicación diferente puede iniciar cualquiera de estas actividades (si la aplicación de correo electrónico lo permite).

Por ejemplo, para que el usuario comparta una imagen, una aplicación de cámara puede iniciar la actividad correspondiente en la aplicación de correo electrónico que redacta el nuevo mensaje. Una actividad posibilita las siguientes interacciones clave entre el sistema y la aplicación:

- Realizar un seguimiento de lo que realmente le interesa al usuario (lo que está en pantalla) para garantizar que el sistema siga ejecutando el proceso que aloja la actividad.
- Saber que los procesos usados con anterioridad contienen elementos a los que el usuario puede regresar (actividades detenidas) y, en consecuencia, priorizar más esos procesos que otros.
- Ayudar a la aplicación a controlar la finalización de su proceso para que el usuario pueda regresar a las actividades con el estado anterior restaurado.
- Permitir que las aplicaciones implementen flujos de usuarios entre sí y que el sistema los coordine (el ejemplo más común es compartir).

Nota: Las actividades se implementan como subclases de la clase Activity.

| | | | |
|---|--|---|---|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | GUIA DE LABORATORIO N° 2 |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |

➤ Servicios

Un servicio es un punto de entrada general que permite mantener la ejecución de una aplicación en segundo plano por diversos motivos.

Es un componente que se ejecuta en segundo plano para realizar operaciones de ejecución prolongada o para realizar tareas de procesos remotos. Un servicio no proporciona una interfaz de usuario. Por ejemplo, un servicio podría reproducir música en segundo plano mientras el usuario se encuentra en otra aplicación, o podría capturar datos en la red sin bloquear la interacción del usuario con una actividad.

Otro componente, como una actividad, puede iniciar el servicio y permitir que se ejecute o enlazarse a él para interactuar. De hecho, existen dos semánticas muy diferentes que los servicios usan para indicarle al sistema cómo administrar una aplicación: Los servicios iniciados le indican al sistema que los siga ejecutando hasta que finalicen su trabajo.

Dos ejemplos serían la sincronización de datos en segundo plano o la reproducción de música después de que el usuario sale de la aplicación.

La sincronización de datos en segundo plano o la reproducción de música también son dos tipos de servicios iniciados muy diferentes que modifican la manera en que el sistema los administra:


- La reproducción de música es algo de lo que el usuario es consciente, por lo que la aplicación se lo comunica al sistema indicándole que quiere ejecutarse en segundo plano y le envía una notificación al respecto al usuario. En este caso, el sistema sabe que debe hacer todo lo posible para mantener el proceso de ese servicio en funcionamiento porque el usuario no estará satisfecho si se interrumpe.
- Un servicio habitual en segundo plano no es algo de lo que el usuario sea consciente, por lo que el sistema tiene más libertad para administrarlo. Puede permitir que se interrumpa (y reiniciarlo posteriormente) si necesita memoria RAM en procesos que son más urgentes para el usuario.

Los servicios enlazados se ejecutan porque otra aplicación (o el sistema) indicó que quiere usarlos. Básicamente, lo que sucede es que un servicio le brinda una **API** a otro proceso. Por lo tanto, el sistema sabe que hay una dependencia entre estos procesos.

En consecuencia, si el proceso A está enlazado a un servicio en el proceso B, sabe que debe mantener funcionando el proceso B (y el servicio correspondiente) para el proceso A. Además, si el proceso A es de interés para el usuario, también sabe que debe tratar el proceso B teniendo esto en cuenta. Gracias a su flexibilidad (o a pesar de ella), los servicios se han convertido en un componente muy útil para todos los tipos de conceptos generales del sistema.

Los fondos de pantalla animados, los receptores de notificaciones, los protectores de pantalla, los métodos de entrada, los servicios de accesibilidad y muchas otras funciones básicas del sistema se compilan como servicios que las aplicaciones implementan y que el sistema vincula cuando deben ejecutarse.

Nota: Un servicio se implementa como una subclase de Service.

| | | | |
|---|--|---|---|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | GUIA DE LABORATORIO N° 2 |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |

➤ **Receptores de emisiones**

Un receptor de emisión es un componente que posibilita que el sistema entregue eventos a la aplicación fuera de un flujo de usuarios habitual, lo que permite que la aplicación responda a los anuncios de emisión de todo el sistema.

Dado que los receptores de emisión son otro punto bien definido de entrada a la aplicación, el sistema puede entregar emisiones incluso a las aplicaciones que no estén en ejecución. Por ejemplo, una aplicación puede programar una alarma para publicar una notificación sobre un evento futuro destinada al usuario. Al entregar dicha alarma al receptor de emisión de la aplicación, no hace falta que dicha aplicación siga ejecutándose hasta que se active la alarma.

Muchas emisiones provienen del sistema (por ejemplo, las que anuncian que se apagó la pantalla, que el nivel de carga de la batería es bajo o que se capturó una imagen). Las aplicaciones también pueden iniciar emisiones, por ejemplo, para avisarles a las demás aplicaciones que se descargaron datos al dispositivo y que están disponibles para el uso.

Si bien los receptores de emisión no exhiben una interfaz de usuario, pueden crear una notificación de la barra de estado para alertar al usuario cuando se produzca un evento de emisión. Sin embargo, por lo general, un receptor de emisión es simplemente una puerta de enlace a otros componentes y está destinado a realizar una cantidad mínima de trabajo.

Por ejemplo, podría programar un servicio **JobService** para que realice algunas tareas en función del evento con **JobScheduler**.

Nota: Un receptor de emisión se implementa como una subclase de **BroadcastReceiver** y cada receptor de emisión se entrega como un objeto Intent.


➤ **Proveedores de contenido**

Un proveedor de contenido administra un conjunto compartido de datos de la aplicación que puedes almacenar en el sistema de archivos, en una base de datos SQLite, en la Web o en cualquier otra ubicación de almacenamiento persistente a la que tenga acceso tu aplicación.

A través del proveedor de contenido, otras aplicaciones pueden consultar o modificar los datos si el proveedor de contenido lo permite. Por ejemplo, el sistema Android proporciona un proveedor de contenido que administra la información de contacto del usuario.

De esta manera, cualquier aplicación con los permisos correspondientes puede consultar el proveedor de contenido (como **ContactsContract.Data**) para la lectura y la escritura de información sobre una persona específica. En ocasiones, se interpreta que el proveedor de contenido es una abstracción en una base de datos, porque tiene un gran volumen de API y compatibilidad incorporado.

Sin embargo, su finalidad principal es diferente desde una perspectiva de diseño del sistema. Para el sistema, un proveedor de contenido es un punto de entrada a una aplicación para publicar elementos de datos con nombre y se identifica mediante un esquema de **URI (identificador de recurso uniforme)**. Así, una aplicación puede

| | | | |
|---|--|---|---|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | GUIA DE LABORATORIO N° 2 |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |

decidir cómo quiere asignar los datos que contiene a un espacio de nombres de URI y entregar esos URI a otras entidades que, a su vez, pueden usarlos para acceder a los datos.

Gracias a esto, el sistema puede realizar algunas tareas específicas cuando administra una aplicación:


- Asignar un URI no exige que la aplicación permanezca ejecutándose, por lo que los URI pueden persistir incluso después de que se cierran las aplicaciones a las que pertenecen. El sistema solo necesita asegurarse de que la aplicación de un URI siga ejecutándose si debe recuperar los datos de la aplicación desde el URI en cuestión.
- Estos URI también ofrecen un modelo de seguridad importante y detallada. Por ejemplo, una aplicación puede colocar el URI de una imagen que tiene en el portapapeles, pero bloquear al proveedor de contenido para que otras aplicaciones no puedan acceder a él libremente. Si otra aplicación intenta acceder a ese URI en el portapapeles, el sistema puede concederle acceso usando un permiso de URI temporal para que solo pueda acceder a los datos mediante ese URI, pero nada más.

Los proveedores de contenido también son útiles para leer y escribir datos privados de tu aplicación y que no se comparten. Un proveedor de contenido se implementa como una subclase de **ContentProvider** y debe implementar un conjunto estándar de API que permitan a otras aplicaciones realizar transacciones.

Un aspecto exclusivo del diseño del sistema Android es que cualquier aplicación puede iniciar un componente de otra aplicación. Por ejemplo, si quieres que el usuario tome una foto con la cámara del dispositivo, probablemente haya otra aplicación para eso y tu aplicación pueda usarla, en lugar de desarrollar una actividad para tomar una foto. No hace falta que incorpores ni vincules la aplicación de la cámara con el código. En cambio, basta con que inicies la actividad en la aplicación de la cámara que captura una foto. Cuando termines, la foto aparecerá en tu aplicación para que puedas usarla. Al usuario le parecerá que la cámara en realidad forma parte de tu aplicación.

Cuando el sistema inicia un componente, inicia el proceso para esa aplicación (si es que no se está ejecutando) y crea una instancia de las clases necesarias para el componente. Por ejemplo, si tu aplicación inicia la actividad en la aplicación de cámara que toma una foto, esa actividad se ejecuta en el proceso que pertenece a la aplicación de cámara, no en el proceso de tu aplicación. Por lo tanto, a diferencia de lo que sucede con las aplicaciones en la mayoría de los demás sistemas, **las aplicaciones de Android no tienen un solo punto de entrada (no existe la función main())**.

Como el sistema ejecuta cada aplicación en un proceso independiente con permisos de archivo que limitan el acceso a otras aplicaciones, tu aplicación no puede activar directamente un componente de otra. Sin embargo, sí puede hacerlo el sistema Android. Para activar un componente en otra aplicación, envía un mensaje al sistema que especifique tu intención de iniciar un componente específico. Luego, el sistema activa ese componente por ti.

| | | | |
|--|--|---|---|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | GUIA DE LABORATORIO N° 2 |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |

➤ Activación de componentes

De los cuatro tipos de componentes, tres (actividades, servicios y receptores de emisión) se activan mediante un mensaje asíncrono denominado **intent**. Las intents vinculan componentes entre sí durante el tiempo de ejecución. Imagina que son los mensajeros que solicitan acciones de otros componentes, ya sea que estos pertenezcan a tu aplicación o a alguna otra.

Una **intent** se crea con un objeto Intent, que define un mensaje para activar un componente específico (intent explícita) o un tipo específico de componente (intent implícita).

Para actividades y servicios, una intent define la acción que se realizará (por ejemplo, ver o enviar algo) y puede especificar el URI de los datos en los que debe actuar, entre otras cosas que el componente que se está iniciando podría necesitar saber.

Por ejemplo, una intent podría transmitir una solicitud para que una actividad muestre una imagen o abra una página web. En algunos casos, puedes iniciar una actividad para recibir un resultado. En ese caso, dicha actividad también devuelve el resultado en una Intent. Por ejemplo, puedes emitir una intent para permitir que el usuario elija un contacto personal y te lo devuelva. Esa intent devuelta incluye un URI dirigido al contacto elegido.

En el caso de los receptores de emisión, la intent tan solo define el anuncio que se emite. Por ejemplo, una emisión para indicar que el nivel de batería del dispositivo es bajo incluye solo una cadena de acción conocida que indica batería baja.


A diferencia de las actividades, los servicios y los receptores de emisión, los proveedores de contenido no se activan con **intents**. En cambio, se activan mediante solicitudes de un **ContentResolver**. El solucionador de contenido aborda todas las transacciones directas con el proveedor de contenido, de modo que el componente que realiza las transacciones con el proveedor no deba hacerlo y, en su lugar, llame a los métodos del objeto ContentResolver. Esto deja una capa de abstracción entre el proveedor de contenido y el componente que solicita información (por motivos de seguridad).

➤ El archivo de manifiesto (AndroidManifest.xml).

Para que el sistema Android pueda iniciar un componente de la aplicación, debe reconocer la existencia de ese componente leyendo el archivo de manifiesto de la aplicación. Tu aplicación debe declarar todos sus componentes en este archivo, que debe encontrarse en la raíz del directorio de proyectos de la aplicación.

El manifiesto puede hacer ciertas cosas además de declarar los componentes de la aplicación, por ejemplo:

- Identificar los permisos de usuario que requiere la aplicación, como acceso a Internet o acceso de lectura para los contactos del usuario.
- Declarar el nivel de API mínimo que requiere la aplicación en función de las API que usa.
- Declarar características de hardware y software que la aplicación usa o exige, como una cámara, servicios de Bluetooth o una pantalla multitáctil.
- Declarar bibliotecas de la API a las que la aplicación necesita estar vinculada (además de las API del marco de trabajo de Android), como la biblioteca de Google Maps.

| | | | |
|---|--|---|---|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | GUÍA DE LABORATORIO N° 2 |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |

➤ Declaración de componentes

La tarea principal del manifiesto es informarle al sistema acerca de los componentes de la aplicación. Por ejemplo, un archivo de manifiesto puede declarar una actividad de la siguiente manera:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:icon="@drawable/app_icon.png" ... >
        <activity android:name="com.example.project.ExampleActivity"
            android:label="@string/example_label" ... >
        </activity>
        ...
    </application>
</manifest>
```

En el elemento <application>, el atributo android:icon señala los recursos de un ícono que identifica la aplicación.

En el elemento <activity>, el atributo android:name especifica el nombre de clase plenamente calificado de la subclase Activity, y el atributo android:label especifica una cadena para usar como etiqueta de la actividad visible para el usuario.

Por cada componente de app que crees en tu aplicación, deberás declarar un elemento XML correspondiente en el archivo de manifiesto:


- **<activity>** para cada subclase de Activity.
- **<service>** para cada subclase de Service.
- **<receiver>** para cada subclase de BroadcastReceiver.
- **<provider>** para cada subclase de ContentProvider.

Si incluyes actividades, servicios y proveedores de contenido en tu archivo de origen, pero no los declaras en el manifiesto, no estarán visibles para el sistema y, por consiguiente, no se podrán ejecutar. No obstante, los receptores de emisión pueden declararse en el manifiesto o crearse dinámicamente en forma de código como objetos **BroadcastReceiver** y registrarse en el sistema llamando a **registerReceiver()**.

➤ Permisos

Las aplicaciones de Android deben solicitar permiso para acceder a datos del usuario confidenciales (como contactos y SMS) o a determinadas funciones del sistema (como la cámara y el acceso a Internet). Cada permiso se identifica con una etiqueta única. Por ejemplo, en el manifiesto de una app que necesite enviar mensajes SMS debe incluirse esta línea:

```
<manifest ... >
    <uses-permission android:name="android.permission.SEND_SMS"/>
    ...
</manifest>
```


| | | | |
|--|--|---|---|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | GUIA DE LABORATORIO N° 2 |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |

A partir de Android 6.0 (nivel de API 23), el usuario puede aprobar o rechazar algunos permisos durante el tiempo de ejecución. No obstante, sin importar qué versión de Android admita tu aplicación, debes declarar todas las solicitudes de permisos con un elemento **<uses-permission>** en el manifiesto. Si se otorga el permiso, la aplicación puede usar las funciones protegidas. De lo contrario, los intentos de acceder a esas funciones fallarán.

Tu aplicación también puede proteger sus propios componentes con permisos. Puede usar cualquiera de los permisos definidos por Android, tal como se indica en **android.Manifest.permission**, o un permiso que esté declarado en otra app. Tu aplicación también puede definir sus propios permisos.

Nota: Un permiso nuevo se declara con el elemento **<permission>**.


V. DESARROLLO DE PRÁCTICA – SEGUNDA PARTE

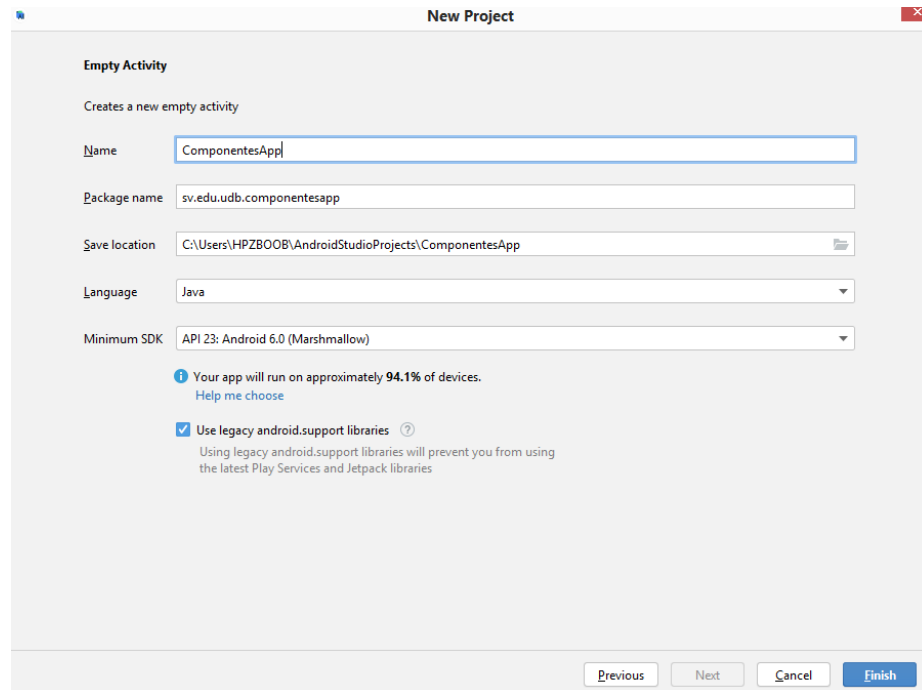
Considere apropiado entender los siguientes conceptos fundamentales en el diseño, durante el desarrollo del ciclo se irán ampliando los temas.

- **wrap_content:** El borde de la vista secundaria se ajusta perfectamente a su propio contenido.
- **match_parent:** El borde de la vista secundaria se expande para coincidir con el borde de la vista principal.
- **LinearLayout:** Es un layout que organiza sus hijos dentro de una fila vertical u horizontal. Crea un scrollbar si el tamaño de la ventana excede el tamaño de la pantalla.
- **RelativeLayout:** Permite especificar la ubicación de los objetos hijos en relación a cada uno o a su padre.
- **ConstraintLayout:** Permite crear diseños grandes y complejos con una jerarquía de vistas plana (sin grupos de vistas anidadas).

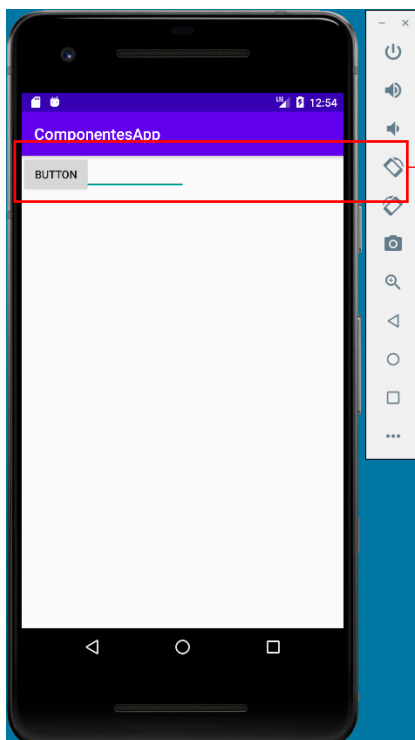
Ejemplo de comunicación entre dos actividades

1. Ejecuta Android Studio y crea un nuevo proyecto. En **Project Template** selecciona **Empty Activity** y presiona **Next**.
2. Como **Name** coloca **ComponentesApp**, en **Package name** escribe **sv.edu.udb.componentesapp** en **Save location** escoge la carpeta de tu preferencia, en Language elige **Java** y el **Minimum SDK** selecciona **API 23: Android 6.0 (Marshmallow)**
3. Presiona **Finish**


| | | | |
|---|--|---|---|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | GUIA DE LABORATORIO N° 2 |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |



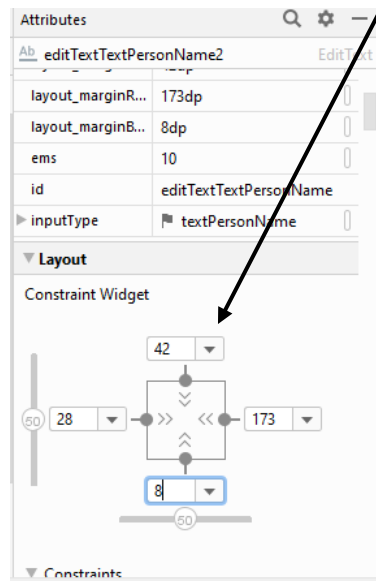
- Comienza a diseñar la activity_main , elimina el TextView inicial, coloca dos Text (Plain Text y Number) y agrega un Button.
- Ejecuta aplicación y observa resultados, todos los controles en una sola posición.



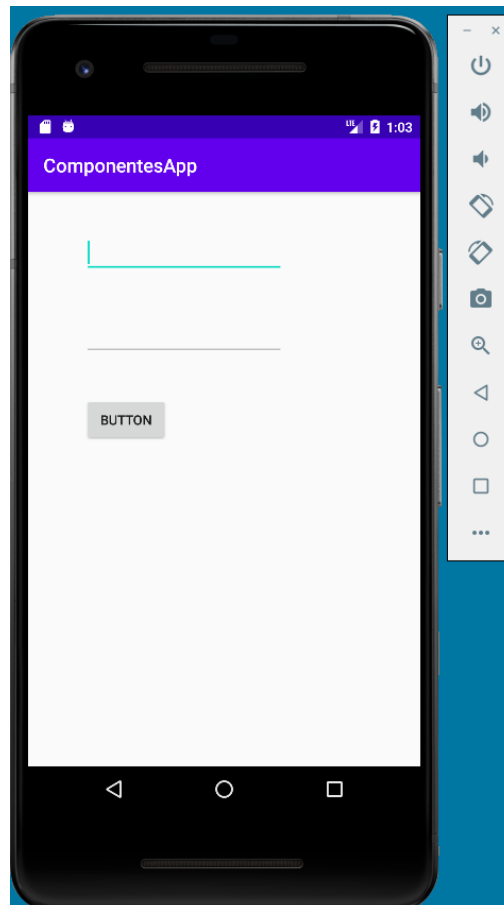
Todos los controles se encuentran en una sola posición.


| | | | |
|---|--|--|---|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | GUIA DE LABORATORIO N° 2 |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |

6. Para resolver el problema **hay varias alternativas**, una es buscar en los atributos la sección **Layout** y dar los márgenes correspondientes a cada control, y así sucesivamente.

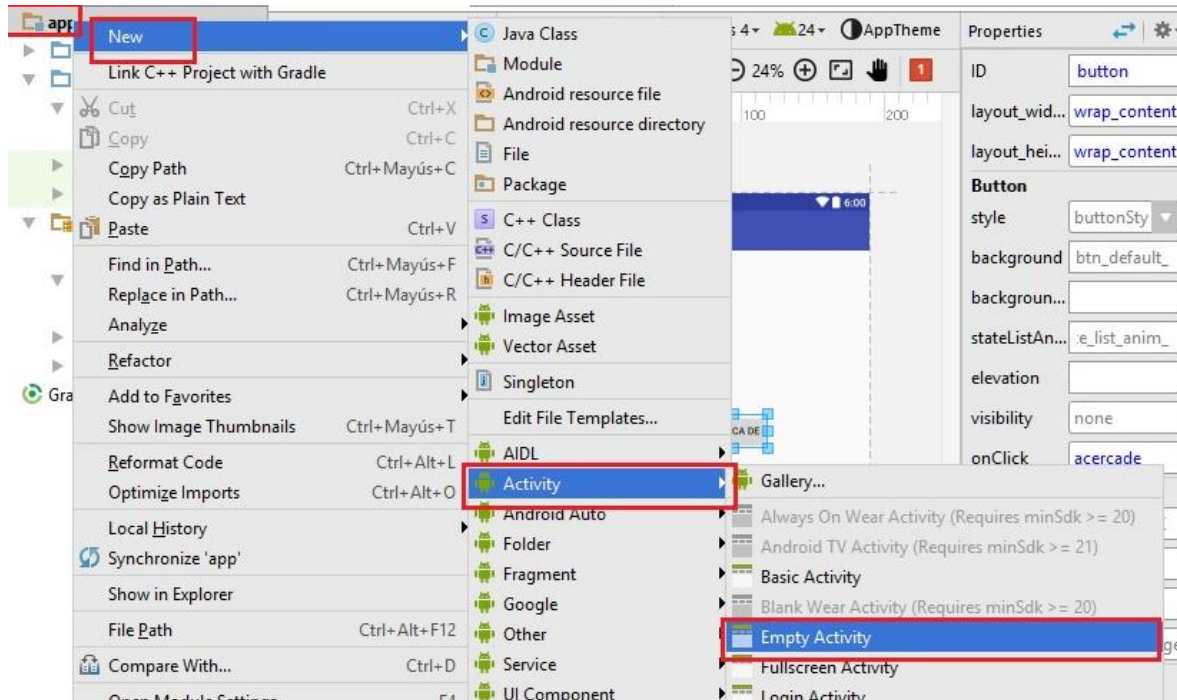


7. Ejecútalo nuevamente, y debe mostrar un resultado similar a la pantalla, los controles ordenados.



| | | | |
|---|--|---|---|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |
| | | | GUIA DE LABORATORIO N° 2 |


8. Agrega una nueva actividad, dando clic derecho en la parte superior del proyecto app, New -> Activity -> Empty activity, coloca el nombre **segundaActividad**



9. Modifica la **activity_main.xml** los atributos de cada control

| Atributo | Text Plan | Text Number | Button |
|----------|-----------------|---------------|------------------|
| id | txtNombre | txtEdad | btnAceptar |
| text | vació | vació | Aceptar |
| hint | Ingresar Nombre | Ingresar Edad | ----- |
| onClick | | | segundaActividad |

10. Abre la clase MainActivity y agrega las siguientes líneas de código:

| | | | |
|---|--|---|---|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | GUIA DE LABORATORIO N° 2 |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |

```

1 package sv.edu.udb.componentesapp;
2
3 import android.content.Intent;
4 import android.support.v7.app.AppCompatActivity;
5 import android.os.Bundle;
6 import android.view.View;
7 import android.widget.EditText;
8
9 public class MainActivity extends AppCompatActivity {
10
11     private EditText etNombre;
12     private EditText etEdad;
13
14     @Override
15     protected void onCreate(Bundle savedInstanceState) {
16         super.onCreate(savedInstanceState);
17         setContentView(R.layout.activity_main);
18
19         etNombre=(EditText)findViewById(R.id.txtNombre);
20         etEdad=(EditText)findViewById(R.id.txtEdad);
21     }
22
23     public void segundaActividad (View v) {
24         Intent i=new Intent( packageContext: this, SegundaActividad.class);
25         i.putExtra( name: "txtNombre", etNombre.getText().toString());
26         i.putExtra( name: "txtEdad", etEdad.getText().toString());
27         startActivity(i);
28     }
29 }


```

Intent: es un objeto de mensajería que puedes usar para solicitar una acción de otro componente de una app. Si bien las intents facilitan la comunicación entre componentes de varias formas.

11. Comience a diseñar la segunda actividad, coloca dos textView y un Button, modifica la **activity_segunda_actividad.xml** los atributos de cada control.

| Atributo | textView 1 | textView 2 | Button |
|----------|----------------|--------------|--------------|
| id | textViewNombre | textViewEdad | btnFinalizar |
| text | ----- | ----- | Finalizar |
| onClick | ----- | ----- | Finalizar |

12. Abre la clase **segundaActividad** y agrega las siguientes líneas de código:


| | | | |
|---|--|--|---|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | GUIA DE LABORATORIO N° 2 |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |

```

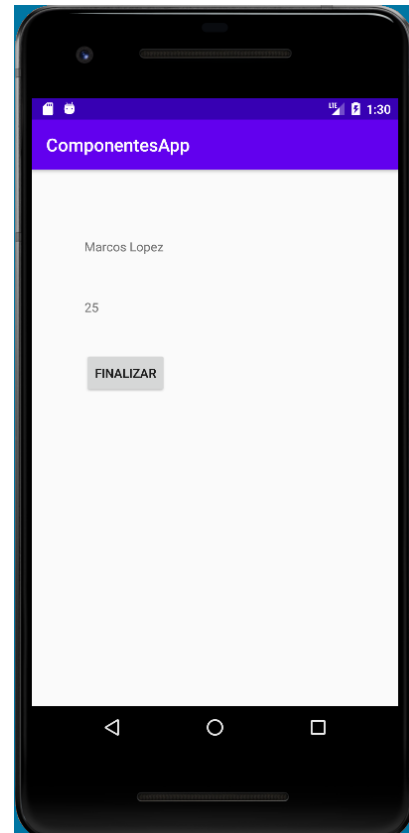
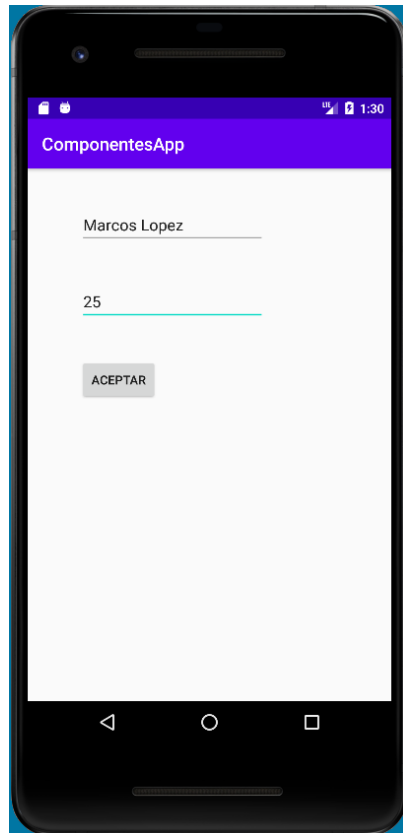
1  package sv.edu.udb.componentesapp;
2
3  import android.support.v7.app.AppCompatActivity;
4  import android.os.Bundle;
5  import android.view.View;
6  import android.widget.TextView;
7
8  public class SegundaActividad extends AppCompatActivity {
9      private TextView tvNombre;
10     private TextView tvEdad;
11
12     @Override
13     protected void onCreate(Bundle savedInstanceState) {
14         super.onCreate(savedInstanceState);
15         setContentView(R.layout.activity_segunda_actividad);
16
17         tvNombre=(TextView)findViewById(R.id.textViewNombre);
18         tvEdad=(TextView)findViewById(R.id.textViewEdad);
19
20         Bundle bundle = getIntent().getExtras();
21         String nombre=bundle.getString( key: "txtNombre");
22         String edad=bundle.getString( key: "txtEdad");
23
24         tvNombre.setText(nombre);
25         tvEdad.setText(edad);
26     }
27
28     public void finalizar(View v) {
29         finish();
30     }
31 }

```

Bundle: Los paquetes se utilizan generalmente para pasar datos entre varias actividades de Android. Depende de qué tipo de valores desee pasar, pero los paquetes pueden contener todo tipo de valores y pasarlos a la nueva actividad

| | | | |
|---|--|---|---|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | GUIA DE LABORATORIO N° 2 |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |


13. Para finalizar deberá mostrar un resultado similar a las pantallas:

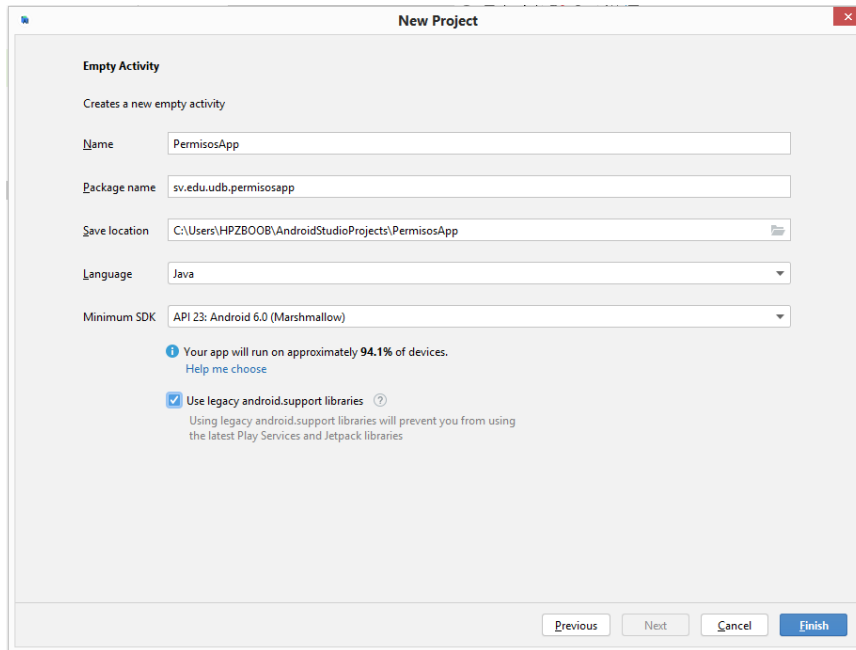


Ejemplo de configuración de permisos

Para este ejemplo se práctica la configuración de permisos para acceder a la agenda de contactos.

1. Ejecuta Android Studio y crea un nuevo proyecto. En **Project Template** selecciona **Empty Activity** y presiona **Next**.
2. Como **Name** coloca **ComponentesApp**, en **Package name** escribe **sv.edu.udb.permisosapp** en **Save location** escoge la carpeta de tu preferencia, en Language elige **Java** y el **Minimum SDK** selecciona **API 23: Android 6.0 (Marshmallow)**
3. Presiona **Finish**

| | | | |
|---|--|---|--------------------------------|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |
| | | GUIA DE LABORATORIO N° 2 | |



4. Agrega en el diseño de la actividad 2 TextView y un Button.
5. Modifica la **activity_main.xml** y modifica los atributos de cada control


| Atributo | TextView | TextView | Button |
|----------|---------------------|-------------|--------------|
| id | lblPermiso | lblContacto | btnContactos |
| text | Estado del permiso: | vacío | Contactos |
| onClick | | | on_Click |

6. Abre el archivo AndroidManifest.xml, aquí concederemos el permiso para acceder a los contactos de nuestro teléfono. Agrega la línea 22 a tu archivo.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="sv.edu.udb.permisosapp">
4
5      <application
6          android:allowBackup="true"
7          android:icon="@mipmap/ic_launcher"
8          android:label="PermisosApp"
9          android:roundIcon="@mipmap/ic_launcher_round"
10         android:supportRtl="true"
11         android:theme="@style/Theme.PermisosApp">
12         <activity
13             android:name=".MainActivity"
14             android:exported="true">
15             <intent-filter>
16                 <action android:name="android.intent.action.MAIN" />
17
18                 <category android:name="android.intent.category.LAUNCHER" />
19             </intent-filter>
20         </activity>
21     </application>
22     <uses-permission android:name="android.permission.READ_CONTACTS" />
23
24 </manifest>

```

| | | | |
|---|--|--|---|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | GUIA DE LABORATORIO N° 2 |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |

7. Utilicemos nuestro archivo de recursos de los valores String correspondiente al archivo res -> values -> strings.xml

```

1  <resources>
2      <string name="app_name">PermisosApp</string>
3      <string name="btnName">Abrir contactos</string>
4      <string name="lblName">Estado del permiso:</string>
5      <string name="lblContacto">Contacto Seleccionado:</string>
6  </resources>

```

8. Definamos las siguientes variables en nuestro archivo MainActivity.java (Nota: recuerda utilizar Alt + Intro, cuando sea necesario)

```

20 public class MainActivity extends AppCompatActivity {
21     private TextView lblPermiso;
22     private TextView lblContacto;
23     private final int MY_PERMISSIONS = 100; // Codigo que utilizamos para solicitar los permisos, puede ser cualquier numero
24
25     private final int OPEN_CONTACT = 200; // Codigo que utilizamos para abrir la lista de contactos, puede ser cualquier numero
26
27     private final String str_permitido = "PERMITIDO";
28     private final String str_denegado = "DENEGADO";
29     private String estado;

```


9. Creemos un método que nos permita verificar los permisos de usuarios, para ello copia el siguiente código en tu archivo MainActivity.java

```

47     public boolean verificarPermiso() {
48         /* Comprobando la version del dispositivo, en este caso que sea la version de Android 6.0 */
49         if (Build.VERSION.SDK_INT < Build.VERSION_CODES.M) return true;
50
51         /*Comprobando que los permisos ya hayan sido aceptados por el usuario*/
52         if (checkSelfPermission(READ_CONTACTS) == PackageManager.PERMISSION_GRANTED)
53             return true;
54
55         // los permisos nos fueron aceptados por el usuario
56         return false;
57     }

```

10. Modifiquemos nuestro método onCreate para que la verificación de los permisos se realice siempre que se inicia la aplicación. Utilizaremos un TextView para mostrar el resultado del estado del permiso. La validación de los permisos se realiza en versiones de Android igual o superior a la 6.0 (Marshmallow).

| | | | |
|---|--|--|---|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | GUIA DE LABORATORIO N° 2 |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |

```

31
32  @Override
33  protected void onCreate(Bundle savedInstanceState) {
34      super.onCreate(savedInstanceState);
35      setContentView(R.layout.activity_main);
36
37      estado = "Estado del permiso:";
38
39      lblPermiso = (TextView) findViewById(R.id.lblPermiso);
40      lblContacto = (TextView) findViewById(R.id.lblContacto);
41
42      if (verificarPermiso())
43          lblPermiso.setText(estado + " " + str_permitido);
44      else
45          lblPermiso.setText(estado + " " + str_denegado);

```

11. Con el siguiente código haremos funcionar nuestro botón “Contactos”, aquí se verificarán los permisos necesarios para acceder a la agenda de contactos.

```


59  @TargetApi(Build.VERSION_CODES.M)
60  public void onClick(View view) {
61      if (verificarPermiso()) {
62          lblPermiso.setText(estado + " " + str_permitido);
63          Intent intent = new Intent(Intent.ACTION_PICK, ContactsContract.Contacts.CONTENT_URI); //Creamos un Intent para abrir los contactos
64          startActivityForResult(intent, OPEN_CONTACT); //Lanzamos el Intent y se espera a que se seleccione un contacto
65      } else {
66          requestPermissions(new String[]{READ_CONTACTS}, MY_PERMISSIONS); //Solicitamos los permisos para abrir los contactos
67      }
68  }

```

startActivityForResult(intent, OPEN_CONTACT): Lanza una actividad con la lista de contactos del teléfono, básicamente abre la aplicación de los contactos, pero con la posibilidad de seleccionar alguno debido al tipo de **Intent** que creamos (**ACTION_PICK**). Al seleccionar un contacto se regresa automáticamente a nuestra aplicación y se ejecuta el método **onActivityResult(int requestCode, int resultCode, Intent data)**, donde el parámetro *int requestCode* se utiliza para identificar a la petición de abrir una actividad, es decir, con este código podemos filtrar y saber lo que se estaba haciendo antes de regresar a nuestra aplicación, en nuestro caso es la variable **OPEN_CONTACT**.

El parámetro **Intent data**, contendrá la información del resultado que estemos esperando con el código que le enviamos, que en nuestro ejemplo será la información del contacto que el usuario seleccione.

12. Por lo tanto procedemos a sobre escribir el método **onActivityResult** en el que de obtendremos el nombre del contacto seleccionado y se lo asignaremos al label “LblContacto” de la siguiente manera:

| | | | |
|---|--|---|---|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | GUIA DE LABORATORIO N° 2 |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |

```

70      @SuppressWarnings("Range")
71      @Override
72      public void onActivityResult(int requestCode, int resultCode, Intent data) {
73          super.onActivityResult(requestCode, resultCode, data);
74
75          if (resultCode == Activity.RESULT_OK) {
76              switch (requestCode) {
77                  case OPEN_CONTACT:
78                      Uri contactUri = data.getData();
79                      // Con el URI consultamos la información del contacto seleccionado
80                      Cursor cursor = getContentResolver().query(contactUri, projection: null, selection: null, selectionArgs: null, sortOrder: null);
81
82                      String nombre = "\n" + "Contacto Seleccionado: " + "\n";
83                      // Verificación si el contacto existe
84                      if (cursor.moveToFirst()) {
85                          // Obtenemos el nombre del contacto
86                          nombre = nombre + cursor.getString(cursor.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME));
87                      }
88                      lblContacto.setText(nombre);
89                      break;
90              }
91          }
92      }
93  }

```

13. Con el método definido por Android


“**requestPermissions(new String[]{*READ_CONTACTS*}, *MY_PERMISSIONS*):**” podremos solicitar uno o varios permisos a la vez, simplemente creando un vector de Strings con el código de los permisos y enviarlo como parámetro. Para saber si los permisos fueron aceptados o no se implementa el método **onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults)**, donde el parámetro **int requestCode** se utiliza para identificar la solicitud del permiso que hemos realizado, en nuestro caso hemos colocado la variable **MY_PERMISSIONS** pero puede ser cualquier valor.

A continuación se muestra la sobre escritura del método **requestPermissions**

```

95      @Override
96      public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
97          super.onRequestPermissionsResult(requestCode, permissions, grantResults);
98
99          if (requestCode == MY_PERMISSIONS) {
100              if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
101                  lblPermiso.setText(estado + " " + str_permitido);
102                  // Creamos el Intent para abrir los contactos
103                  Intent intent = new Intent(Intent.ACTION_PICK, ContactsContract.Contacts.CONTENT_URI);
104                  //Abrimos los contactos
105                  startActivityForResult(intent, OPEN_CONTACT);
106              } else {
107                  lblPermiso.setText(estado + " " + str_denegado);
108              }
109          }
110      }
111  }

```

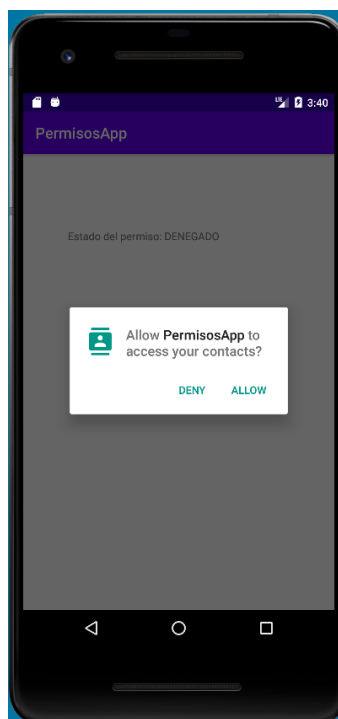
| | | | |
|---|--|---|---|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | GUIA DE LABORATORIO N° 2 |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |


14. Una vez realizado la codificación necesaria procedemos a iniciar nuestra aplicación. Tendríamos que visualizar en nuestra aplicación la siguiente pantalla:

Observe que el estado del permiso se encuentra en Denegado

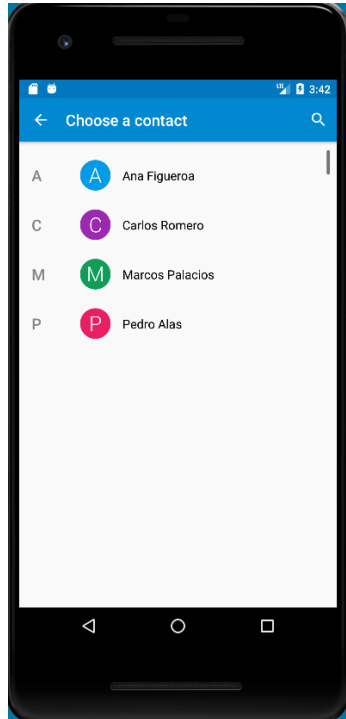


15. Pulse el botón Contactos, tendría que visualizar el cuadro de dialogo para conceder permisos. Acepte los permisos y podrá acceder a la agenda de contactos.

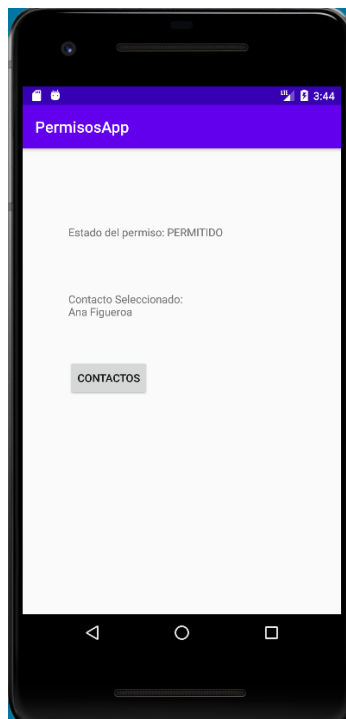



| | | | |
|---|--|---|---|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | GUIA DE LABORATORIO N° 2 |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |

16. Proceda a seleccionar un contacto de su agenda (en este punto debería de tener contactos creados).



17. Observara que el nombre de su contacto se muestra en la aplicación y adicionalmente el estado del permiso ha cambiado.



| | | | |
|---|--|---|---|
|  | UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN | | CICLO 01-2022 |
| | MATERIA | DESARROLLO DE SOFTWARE PARA MÓVILES | GUIA DE LABORATORIO N° 2 |
| | PRÁCTICA | CICLO DE VIDA, COMPONENTES Y PERMISOS DE UNA APP | |

VI. 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**.
2. Escribir un programa en Android, que me permita calcular las deducciones de salarios, donde muestre dos actividades, la primera solicite los datos (nombre, horas trabajadas) internamente realice las operaciones de calcular el salario, conociendo que cada hora equivale a \$8.50 y las deducciones son, ISSS 2%, AFP 3%, RENTA 4%, muestre el detalle con los descuentos establecidos en la segunda actividad. (Salario Neto, ISSS, AFP, RENTA).

VII. BIBLOGRAFÍA

- Documentación Oficial Android <https://developer.android.com/guide/components/activities/activity-lifecycle>
- ViewModel <https://developer.android.com/topic/libraries/architecture/viewmodel>
- developer.android. (s. f.). Aspectos fundamentales de la aplicación. Recuperado 6 de febrero de 2021, de <https://developer.android.com/guide/components/fundamentals?hl=es-419>
- AndroidManifest.xml, <https://developer.android.com/guide/topics/manifest/manifest-intro?hl=es-419>