	UNIVERSIDAD DON BOSCO **FACULTAD DE INGENIERÍA** **ESCUELA DE COMPUTACIÓN**	
CICLO: I/2021	**GUIA DE LABORATORIO #04**	
	Nombre de la Práctica:	Guía #04- Componentes y Permisos
	MATERIA:	Desarrollo de Software para Móviles

I. OBJETIVOS

1. Conocer los tipos de componentes en Android
2. Crear una aplicación en Android, donde se comuniquen dos Actividades.
3. Conocer el archivo manifiesto, para los permisos en Android

II. INTRODUCCION TEORICA

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.

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.

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**.
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 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 detallado. 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. Para obtener más información, consulta la guía para desarrolladores Proveedores de contenido.

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()).**

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.

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>
```

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.

IV. PROCEDIMIENTO

Antes de iniciar algunos 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).

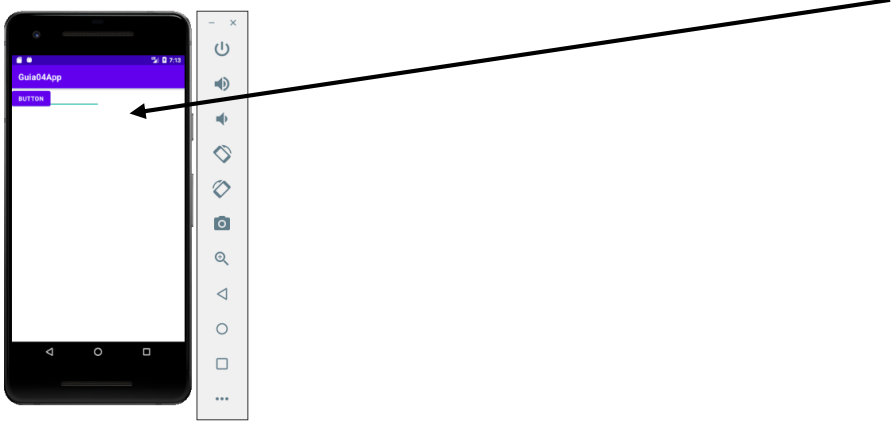
Este ejemplo se practica la interfaz gráfica y la comunicación entre dos actividades.

1. Ejecuta Android Studio y crea un nuevo proyecto. En Project Template selecciona Empty Activity y presiona Next.

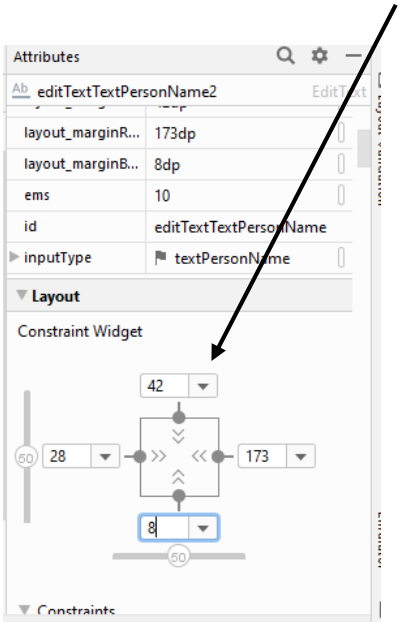
2. Como nombre **Guia04App**, en Package name escribe **sv.edu.udb.guia04app** en Save location escoge la carpeta de tu preferencia, en Language elige Java y el Minimum SDK selecciona API 16: Android 4.1 (Jelly Bean) y Presiona Finish.

3. Comienza a diseñar la activity_main , antes elimina el TextView inicial, coloca dos Text (Plain Text y Number) limpiar el atributo text y además agregar un Button

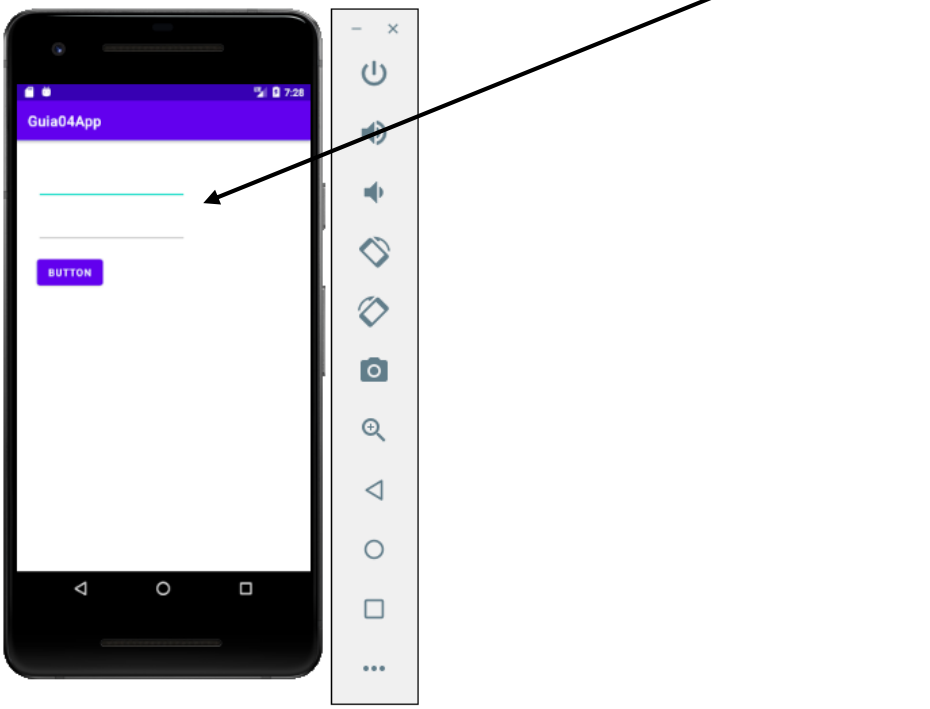
4. Ejecuta aplicación y observa resultados, todos los controles en una sola posición.



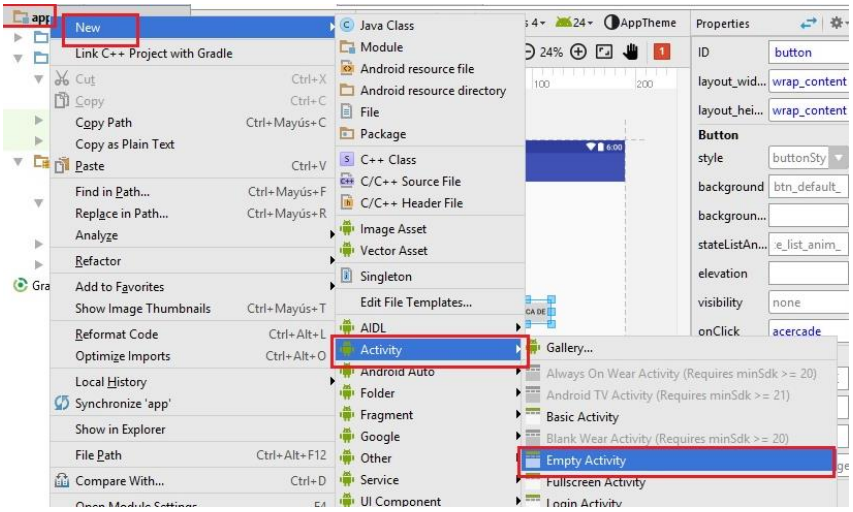
5. Para resolver el problema **hay varias alternativas**, una es buscar en los atributos la sección **Layout** y dar los márgenes correspondientes a cada control, y así sucesivamente.



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



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



8. 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

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

```
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.content.Intent;

public class MainActivity extends AppCompatActivity {

    private EditText etNombre;
    private EditText etEdad;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        etNombre=(EditText)findViewById(R.id.txtNombre);
        etEdad=(EditText)findViewById(R.id.txtEdad);
    }
```

```
    public void segundaActividad (View v) {
        Intent i=new Intent(this, segundaActivida.class);
        i.putExtra("txtNombre", etNombre.getText().toString());
        i.putExtra("txtEdad", etEdad.getText().toString());
        startActivity(i);
    }
}
```

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.

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

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

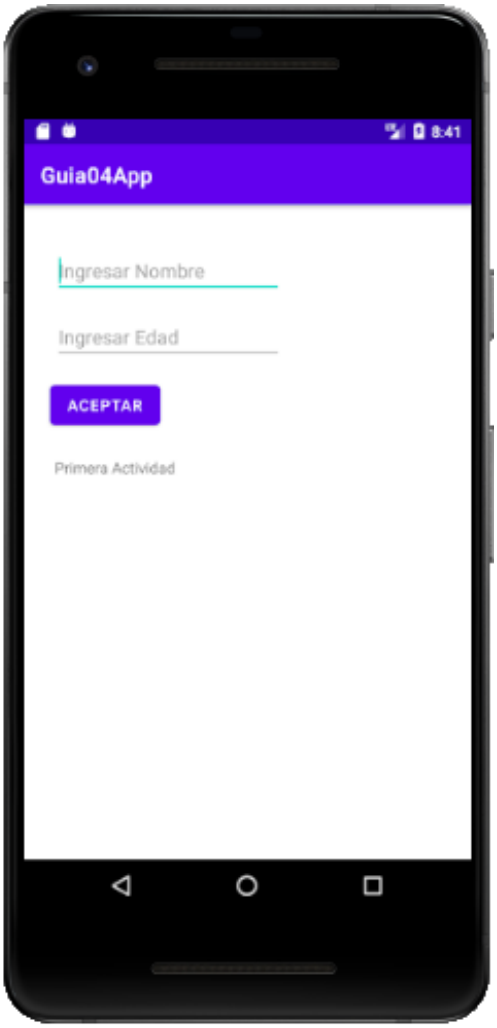
11. Abre la clase **segundaActivida** y agrega las siguientes líneas de código:

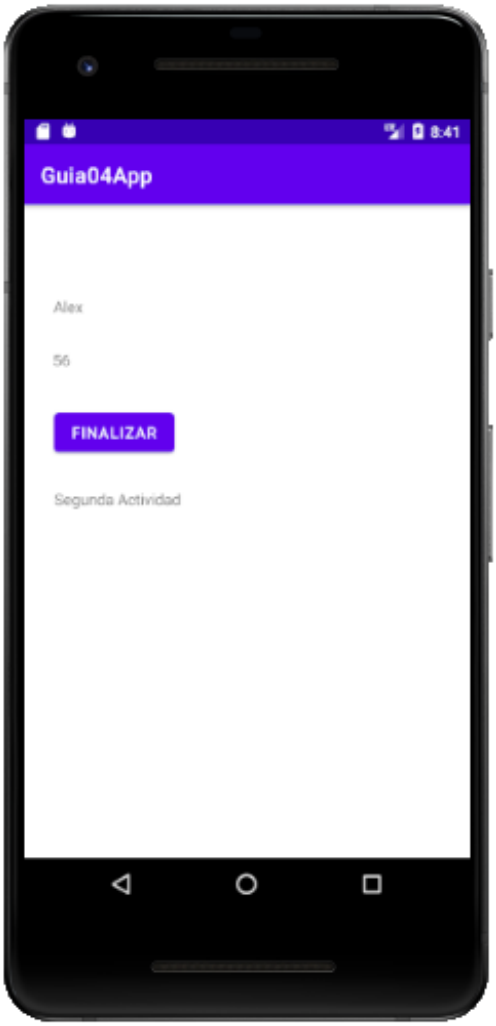
```
package sv.edu.udb.guia04app;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class segundaActivida extends AppCompatActivity {

    private TextView tvNombre;
    private TextView tvEdad;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_segunda_activida);

        tvNombre=(TextView)findViewById(R.id.textViewNombre);
        tvEdad=(TextView)findViewById(R.id.textViewEdad);

        Bundle bundle = getIntent().getExtras();
        String nombre=bundle.getString("txtNombre");
        String edad=bundle.getString("txtEdad");

        tvNombre.setText(nombre);
        tvEdad.setText(edad);
    }
```

```
    public void finalizar(View v) {
        finish();
    }
}
```

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

12. Para finalizar deberá mostrar un resultado similar a las pantallas





V. DISCUSION DE RESULTADOS

1. 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. BIBLIOGRAFIA

- 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