Cómo iniciar otra actividad

Luego de terminar la primera parte de la actividad, tendrás una app que mostrará una actividad que consiste en una sola pantalla con un campo de texto y un botón **Enviar**.
Ahora, podrás agregar código a **MainActivity** con el objetivo de iniciar una nueva actividad para mostrar un mensaje cuando el usuario presione el botón **Enviar**.

Nota: En esta lección, se asume que usas Android Studio 3.0 o una versión posterior.
Cómo responder al botón Enviar

Sigue estos pasos para agregar un método a la clase **MainActivity**, a la que se llama cuando presionas el botón **Enviar**:

1. En el archivo **app > java > com.example.myfirstapp (nombre del archivo y paquete) > MainActivity**, agrega el stub del siguiente método **sendMessage()**:

Java

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    /** Llamado cuando el usuario toca el botón Enviar  */
    public void sendMessage(View view) {
        // Do something in response to button
    }
}
```

Es posible que veas un error porque Android Studio no puede resolver la clase **View** que se usa como argumento del método. Para borrar el error, haz clic en la declaración **View**, coloca el cursor sobre ella y presiona Alt + Intro a fin de realizar una corrección rápida. Si aparece un menú, selecciona **Import class**.

2. Regresa al archivo **activity_main.xml** para llamar al método desde el botón, de la siguiente manera:
    1. Selecciona el botón en el editor de diseño.
    2. En la ventana **Attributes**, ubica la propiedad **onClick** y selecciona **sendMessage [MainActivity]** de la lista desplegable.
       De esta manera, cuando se presione el botón, el sistema llamará al método **sendMessage()**.
       Toma nota de los detalles de este método. Son necesarios para que el sistema reconozca el método como compatible con el atributo **android:onClick**. Específicamente, el método tiene las siguientes características:
    3. Acceso público
2. Un valor que se muestra de unit vacío
3. Un objeto **View** como único parámetro, que es el objeto **View** en el que hiciste clic al final del paso1
3. A continuación, deberás completar este método para leer el contenido del campo de texto y proporcionar dicho texto a otra actividad

Cómo compilar un intent

Un **Intent** es un objeto que proporciona vinculación en tiempo de ejecución entre componentes separados, como dos actividades. El **Intent** representa la intención que tiene una app de realizar una tarea.
Puedes usar intents para varias tareas; aqui, tu intent inicia otra actividad.

En **MainActivity**, agrega la constante **EXTRA_MESSAGE** y el código **sendMessage()**, como se muestra a continuación:

Java

```
public class MainActivity extends AppCompatActivity {
    public static final String EXTRA_MESSAGE =
"com.example.myfirstapp.MESSAGE";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
  /** Llamado cuando el usuario toca el botón Enviar  */
    public void sendMessage(View view) {
        Intent intent = new Intent(this, DisplayMessageActivity.class);
        EditText editText = (EditText)
findViewById(R.id.editTextTextPersonName);
        String message = editText.getText().toString();
        intent.putExtra(EXTRA_MESSAGE, message);
        startActivity(intent);
    }
}
```

Puedes esperar que Android Studio vuelva a mostrar errores **Cannot resolve symbol**.
Para borrar los errores, presiona Alt + Intro.
Deberás realizar las siguientes importaciones:

Java

```
import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
```

Todavía queda un error para **DisplayMessageActivity**, pero no te preocupes porque lo solucionarás en el siguiente paso.
Esto es lo que sucede en **sendMessage()**:

- El constructor **Intent** toma dos parámetros, un **Context** y una **Class**.
    - El parámetro **Context** se usa primero porque la clase **Activity** es una subclase de **Context**.
    - El parámetro **Class** del componente de la app, al que el sistema entrega el **Intent,**, es, en este caso, la actividad que va a comenzar.
- El método **putExtra()** agrega el valor de **EditText** al intent.

- Un **Intent** puede transportar tipos de datos como pares clave-valor llamados *extras*.
  Tu clave es una constante pública **EXTRA_MESSAGE** porque la actividad siguiente usa la clave para obtener el valor de texto.
  Te recomendamos definir las claves para extras de intents con el nombre del paquete de la app como prefijo. De esta manera, se garantiza que las claves sean únicas en el caso de que tu app interactúe con otras aplicaciones.

- El método **startActivity()** inicia una instancia de **DisplayMessageActivity** que especifica el **Intent**. Luego, deberás crear esa clase.

  Nota: El componente de la arquitectura de navegación te permite usar el editor de navegación para asociar una actividad con otra. Una vez que se establezca la relación, podrás usar la API para iniciar la segunda actividad cuando el usuario active la acción asociada, por ejemplo, cuando haga clic en un botón.

Cómo crear la segunda actividad

Para crear la segunda actividad, sigue estos pasos:

1. En la ventana **Project**, haz clic con el botón derecho en la carpeta de la **app** y selecciona **New > Activity > Empty** Activity.
2. En la ventana **Configure Activity**, ingresa "DisplayMessageActivity" en **Activity Name**. Deja todas las demás propiedades con sus valores predeterminados y haz clic en **Finish**.
   Android Studio realiza tres acciones automáticamente:

- Crea el archivo **DisplayMessageActivity**.
- Crea el archivo de diseño **activity_display_message.xml**, que se corresponde con el archivo **DisplayMessageActivity**.
- Agrega el elemento **<activity>** obligatorio en **AndroidManifest.xml**.
  Si ejecutas la app y presionas el botón en la primera actividad, se iniciará la segunda. Sin embargo, estará vacía porque usa el diseño vacío proporcionado por la plantilla.

Cómo agregar una vista de texto

La nueva actividad incluye un archivo de diseño en blanco. Sigue estos pasos para agregar una vista de texto al lugar donde aparece el mensaje:

1. Abre el archivo **app > res > layout > activity_display_message.xml**.

2. Haz clic en **Enable Autoconnection to Parent**  en la barra de herramientas. De esta manera, se habilita la conexión automática. Consulta la figura 1.

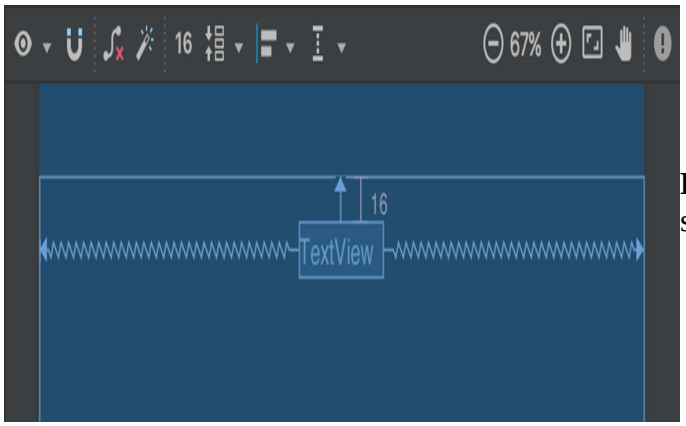

Figura 1: Vista de texto centrada en la parte superior del diseño

3. En el panel **Palette**, haz clic en **Text**, arrastra un elemento **TextView** al diseño y suéltalo cerca de la parte central superior del diseño de manera que se ajuste a la línea vertical que aparecerá. La opción de conexión automática agrega restricciones a la derecha y a la izquierda para ubicar la vista en el centro horizontal.
4. Crea una restricción más desde la parte superior de la vista de texto hasta la parte superior del diseño para que se vea como en la figura 1.

   De manera opcional, puedes realizar algunos ajustes en el estilo de texto si expandes **textAppearance** en el panel **Common Attributes** de la ventana **Attributes** y cambias los atributos como **textSize** y **textColor**.

Cómo mostrar el mensaje

En este paso, puedes modificar la segunda actividad para mostrar el mensaje que pasó la primera actividad.

1. En **DisplayMessageActivity**, agrega el siguiente código al método **onCreate()**:

Java

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

    //Obtenga el intent que inició esta actividad y extraiga la cadena
    Intent intent = getIntent();
    String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
    // Capture el TextView y establezca la cadena como su texto
    TextView textView = findViewById(R.id.textView);
    textView.setText(message);
}
```

2. Presiona Alt + Intro
3. Importar estas otras clases necesarias:

Java

```
import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.TextView;
```

Cómo agregar navegación ascendente

En cada pantalla de tu app que no sea el punto de entrada principal, es decir, todas las pantallas que no sean la **"pantalla principal"**, se debe proporcionar navegación para que el usuario pueda regresar a la pantalla superior lógica en la jerarquía de la app.

- Para ello, agrega un botón **Arriba** en la barra de la app.
- Para agregar un botón **Arriba**, debes declarar qué actividad es la superior lógica en el archivo **AndroidManifest.xml**.

- Abre el archivo que se encuentra en **app** > **manifests** > **AndroidManifest.xml**, busca la etiqueta <**activity**> para **DisplayMessageActivity** y reemplázala por lo siguiente:

```
<activity android:name=".DisplayMessageActivity"
            android:parentActivityName=".MainActivity">
    <!-- la etiqueta meta-data tag si requiere una version de API 15 o menor -->
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity" />
</activity>
```

El sistema Android agregará automáticamente el botón **Arriba** en la barra de la app.

Ejecuta la app

Haz clic en **Apply Changes** en la barra de herramientas para ejecutar la app.
Cuando se abra, escribe un mensaje en el campo de texto y presiona **Send** a fin de ver el mensaje en la segunda actividad.

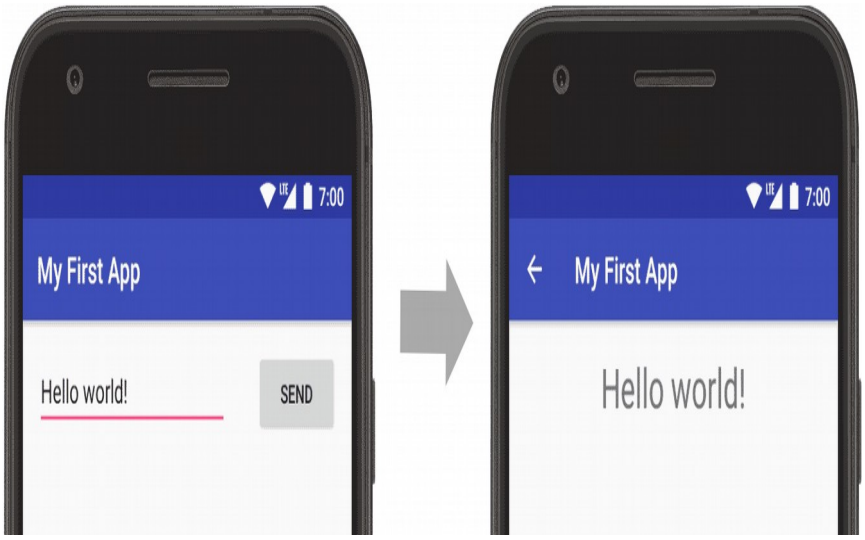

Figura 2: Imagen de la app abierta, con texto que se ingresa a la izquierda de la pantalla y se muestra a la derecha