Guía de ejercicios - Elementos de la interfaz, navegación e interacción (III)

¡Hola! Te damos la bienvenida a esta nueva guía de estudio.

¿En qué consiste esta guía?

La siguiente guía de estudio tiene como objetivo practicar y ejercitar los contenidos que hemos visto en clase.

¡Vamos con todo!

Tabla de contenidos

¡Comencemos!

Actividad guiada N° 1: ¡Texto oculto!

1. Crear un nuevo proyecto desde el template de Empty Activity.

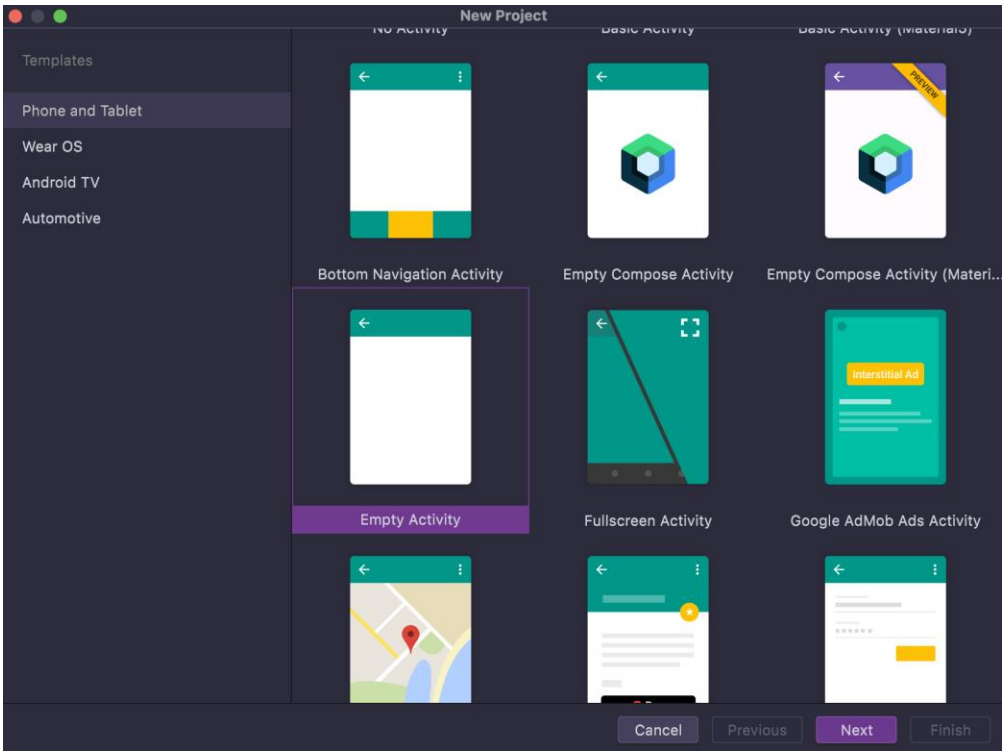


Imagen 01. Paso 1.
Fuente: ADL.

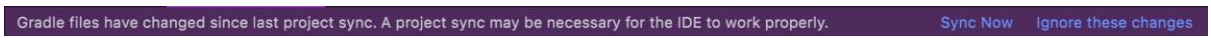
2. Necesitamos acceder al EditText y al Button para asignarles los listeners. Para poder accederlos de forma fácil, vamos a activar ViewBinding, agregando el apartado de `buildFeatures` en el archivo build.gradle.

```
android {
    compileSdk 32

    defaultConfig { ... }

    buildFeatures {
        viewBinding true
    }
}
```

No olvides que siempre que se modifica el archivo build.gradle es necesario sincronizar el proyecto



3. Modificamos el layout activity_main.xml con los componentes necesarios: un EditText de material design, un botón (Button) y un texto oculto (TextView).

Inicialmente, el botón "SHOW MESSAGE" está desactivado, indicando su propiedad `android:enabled="false";` y no reacciona a ningún evento de entrada del usuario. Además, el texto debajo del botón está invisible, indicando su propiedad `android:visibility="invisible"`

Imagen 02. Botón desactivado.
Fuente: ADL.

Desglosando la pantalla anterior, el layout está compuesto por 3 elementos:

- El TextInputLayout que contiene un TextInputEditText para que se introduzca la contraseña.
    - El TextInputLayout permite mostrar un *hint* con un texto descriptivo del objetivo del TextInputEditText.
- El botón (Button) con el mensaje "SHOW MESSAGE", inicialmente visible y desactivado.
- El mensaje (TextView) para mostrar debe estar inicialmente invisible.

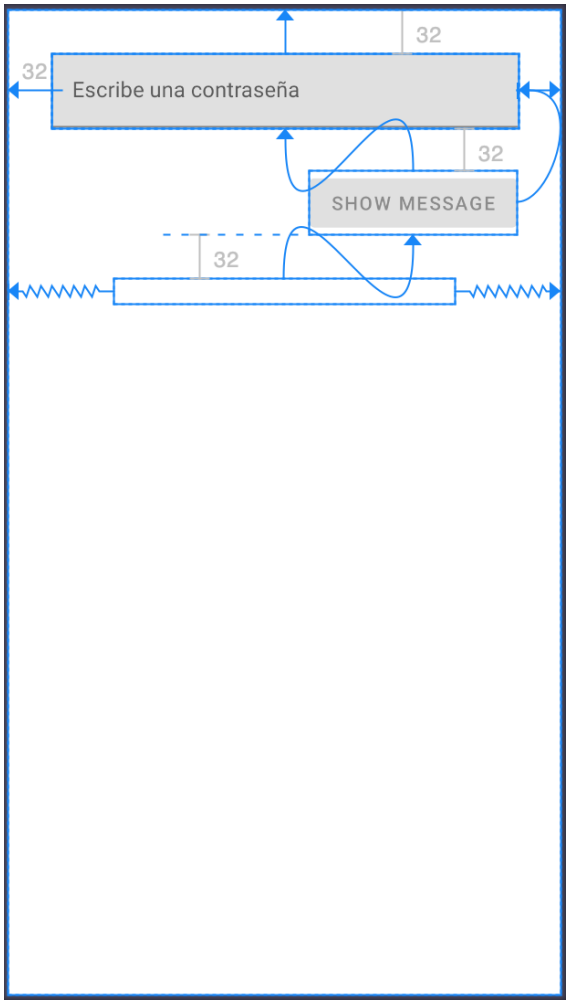


Imagen 03. Botón desactivado.
Fuente: ADL.

En el layout activity_main.xml se ubican los elementos usando un ConstraintLayout que anida los 3 elementos definidos.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".MainActivity">

   <com.google.android.material.textfield.TextInputLayout
       android:id="@+id/passwordInputLayout"
       android:layout_width="0dp"
       android:layout_height="wrap_content"
       android:layout_marginStart="32dp"
       android:layout_marginTop="32dp"
       android:layout_marginEnd="32dp"
```

```
        android:hint="Escribe una contraseña"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <com.google.android.material.textfield.TextInputEditText
            android:id="@+id/passwordEditText"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:inputType="textPassword" />
    </com.google.android.material.textfield.TextInputLayout>

    <Button
        android:id="@+id/showButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="32dp"
        android:enabled="false"
        android:text="Show Message"
        app:layout_constraintEnd_toEndOf="@+id/passwordInputLayout"
        app:layout_constraintTop_toBottomOf="@+id/passwordInputLayout" />

    <TextView
        android:id="@+id/hiddenText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="32dp"
        android:text="La contraseña tiene más de 5 caracteres"
        android:visibility="invisible"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/showButton" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

¿Qué sucede con los textos? Están definidos en este archivo para facilitar la lectura, pero recuerda que **SIEMPRE** deben estar declarados en res/values/strings.xml para poder entregar una mejor experiencia de usuario.

4. Declaramos el atributo para la binding class en MainActivity, modificando el método `onCreate()` para instanciar la binding class y aplicar la vista usando `setContentView(...)` directamente desde el atributo `binding`.

```
public class MainActivity extends AppCompatActivity {

    ActivityMainBinding binding;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        binding = ActivityMainBinding.inflate(getLayoutInflater());
```

```
        setContentView(binding.getRoot());
    }
}
```

5. Creamos el *change listener* para escuchar los cambios en el EditText.
   El método `onTextChanged()` es llamado cuando cambia el texto, por lo que validamos ahí su extensión actual introducida, y activamos el botón cambiando su atributo enabled.

   Desde el método **`onCreate`**() se invoca al método registerListener que agrega un nuevo `TextWatcher()` para reaccionar cuando se cambie el contenido del EditText.

```
public class MainActivity extends AppCompatActivity {

    ActivityMainBinding binding;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        . . .
        registerListener();
    }

    private void registerListener() {
        binding.passwordEditText.addTextChangedListener(new TextWatcher() {
            @Override
            public void beforeTextChanged(CharSequence cs, int i, int i1, int i2) {
            }

            @Override
            public void onTextChanged(CharSequence charSequence, int i, int i1, int i2) {
                if(charSequence.length() > 5) {
                    binding.showButton.setEnabled(true);
                }
            }

            @Override
            public void afterTextChanged(Editable editable) {
            }
        });
    }
}
```

6. Agregamos al método **`registerListener`**() un nuevo listener del tipo OnClickListener que utilizará el botón para mostrar el texto oculto, cambiando la visibilidad a View.VISIBLE.

```
public class MainActivity extends AppCompatActivity {

   ActivityMainBinding binding;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       . . .
   }

   private void registerListener() {
       binding.passwordEditText.addTextChangedListener(new TextWatcher() {
           @Override
           public void beforeTextChanged(CharSequence cs, int i, int i1, int i2) {

           }

           @Override
           public void onTextChanged(CharSequence charSequence, int i, int i1, int i2) {
               if(charSequence.length() > 5) {
                   binding.showButton.setEnabled(true);
               }
           }

           @Override
           public void afterTextChanged(Editable editable) {

           }
       });

       binding.showButton.setOnClickListener(view -> showMessage());
   }

   private void showMessage() {
       binding.hiddenText.setVisibility(View.VISIBLE);
   }
}
```

7. Al correr la app y escribir más de 5 caracteres en el campo de texto, vemos que se activa el botón SHOW MESSAGE. Al presionar el botón, se hace visible el texto “La contraseña tiene más de 5 caracteres”.

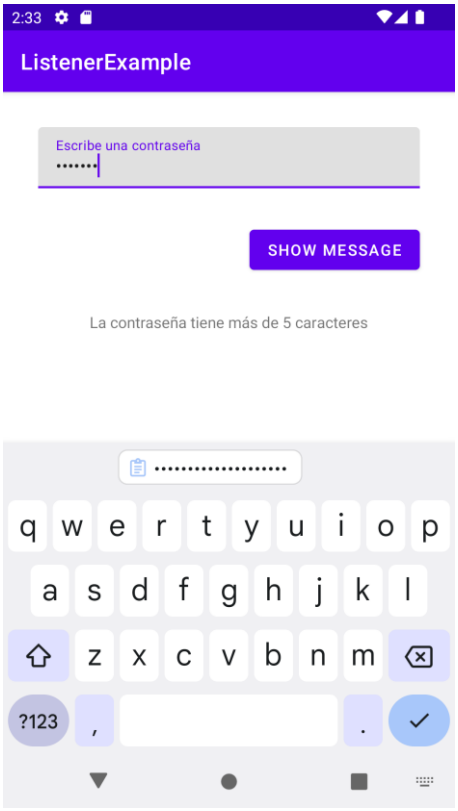


Imagen 04. Show Message.
Fuente: ADL.

¡Manos a la obra! - Habilitar / Deshabilitar el botón SHOW MESSAGE

En nuestro ejemplo, el botón SHOW MESSAGE se activa luego de escribir una contraseña con más de 5 caracteres, pero ¿qué pasa con el botón cuando se eliminan caracteres y ya NO se cumple con la regla?

Implementa el comportamiento esperado en este caso. Si hay menos de 5 caracteres en el campo de texto, el botón debe estar deshabilitado.

¡Manos a la obra! - Usar 7 caracteres

En nuestro ejemplo establecimos 5 caracteres en forma arbitraria, ahora deberás modificar el código para que el mínimo sean 7 caracteres. Recuerda cambiar la validación y los textos asociados.

Tip: Cuando se ocupa la misma información en distintos lugares del código, es buena opción definir una variable con el valor y luego utilizar esa variable en los lugares requeridos, de esa forma la modificación es centralizada.

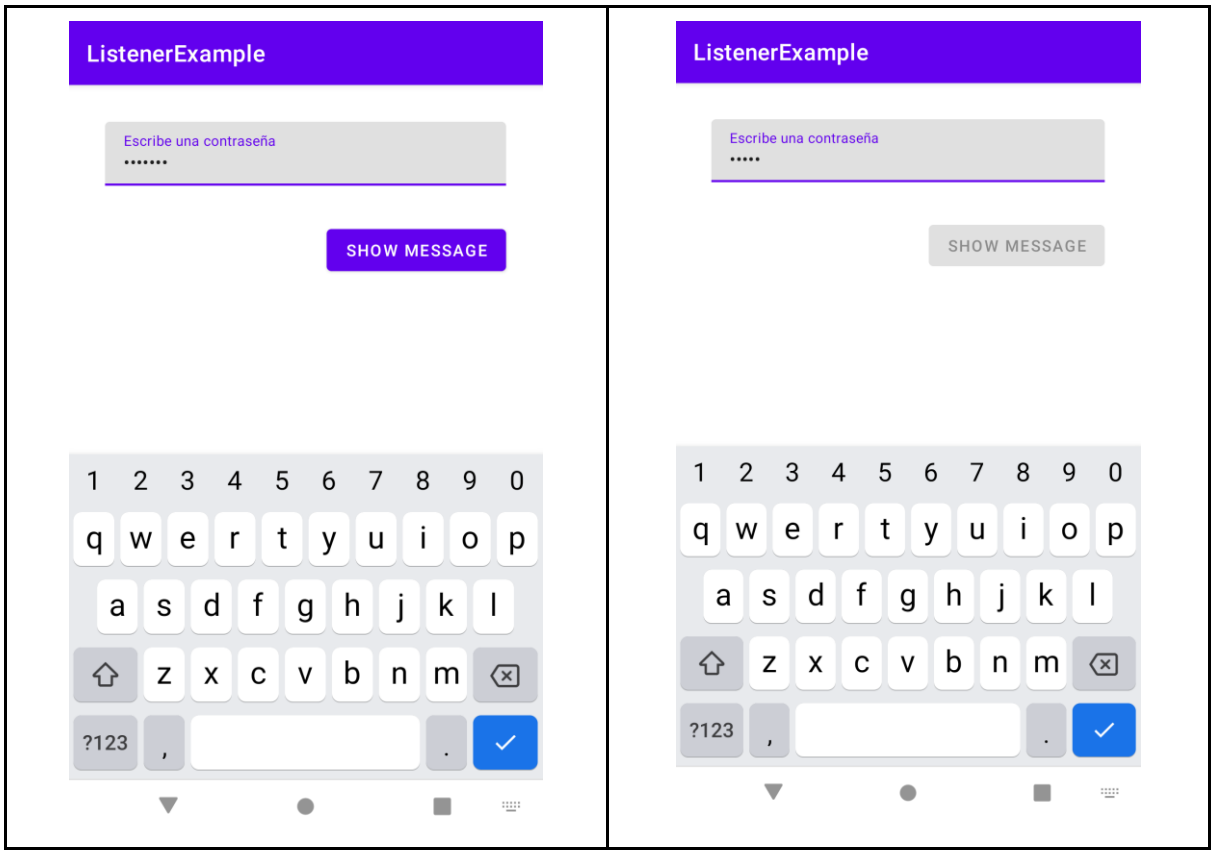
Preguntas de cierre

- ¿En qué casos es necesario agregar un listener a una vista?
- Si queremos que una vista reaccione al clic y un clic largo sobre una imagen:
  - ¿Qué tipos de listener hay que agregar?
  - ¿Cuántos listener hay definir?
- ¿Qué falta repasar?

Actividad complementaria

Define los textos del botón y de los mensajes a mostrar en el archivo res/values/strings.xml para que sean consistentes entre sí.

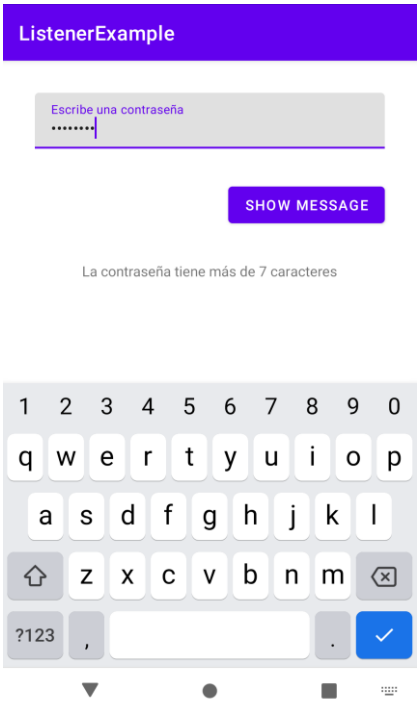
Solución “Manos a la obra” 1



El TextWatcher reacciona a los cambios del texto, por lo tanto, podemos habilitar/deshabilitar el botón al reaccionar en el método onTextChanged()

```
@Override
public void onTextChanged(CharSequence charSequence, int i, int i1, int i2) {
   binding.showButton.setEnabled(charSequence.length() > 5);
}
```

Solución “Manos a la obra” 2



Se puede modificar el valor directamente en el método onTextChanged(), pero se recomienda que este tipo de valores estén definidos en una variable que no se pueda modificar.
Agregamos atributo al fragmento o actividad

```
private static final int MIN_CHARACTERS = 7;
```

y lo utilizamos en el método onTextChanged() para realizar la validación

```
@Override
public void onTextChanged(CharSequence charSequence, int i, int i1, int i2) {
   binding.showButton.setEnabled(charSequence.length() >= MIN_CHARACTERS);
}
```

Solución actividad complementaria