



Asignatura: Desarrollo de Aplicaciones Móviles Nativas.

Tema: El campo de texto `EditText`.

Introducción.

Un componente `EditText` consiste en una interfaz de usuario para introducir, obtener y modificar texto. Muestra un campo de texto con características especificadas en los atributos de su etiqueta `EditText` en el archivo XML.

La clase `EditText` tiene la siguiente jerarquía:

```
java.lang.Object
└─ android.view.View
    └─ android.widget.TextView
        └─ android.widget.EditText
```

Es decir: `public class EditText extends TextView.`

Algunos atributos básicos de un `EditText` son los siguientes:

```
<EditText
    android:id = "@+id/miEditText"
    android:layout_width = "match_parent"
    android:layout_height = "wrap_content"
    android:textColor = "#f00"
    android:hint = "Ingrese su nombre aquí"
    android:textColorHint = "#0f0"
    android:textSize = "25sp"
    android:textStyle = "bold|italic"
    android:text = "Tu correo electrónico"
    android:layout_centerInParent = "true"
    android:padding = "15dp"
    android:background = "#000"
    android:inputType="textPassword"
    android:gravity = "right" /> <!-- gravity la alinea el texto en el campo -->
```

Desarrollo.

Ejecutar cada uno de los siguientes ejercicios y obtener la captura de la pantalla correspondiente del dispositivo, virtual o real.

La clase `EditText`.

1. Un componente `EditText`.

El componente `EditText` es similar a un `TextView` que permite la edición de su contenido para escribir el texto ingresado por el usuario. Este componente muestra una línea inferior con el color del acento del tema y un hint que representa el texto auxiliar asociado.

Generar un nuevo proyecto con una clase `MainActivity.java` y una plantilla `activity_main.xml`. Abrir el archivo `activity_main.xml` en el editor de Android Studio y agregar el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <EditText
        android:id="@+id/xet1"
        android:layout_width="match_parent"
```



```

        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:hint="Texto de entrada" />
</RelativeLayout>

```

El archivo **MainActivity.java**. El paquete inicial no se indica porque depende de la ubicación del proyecto en particular.

```

// Archivo: MainActivity.java
import android.app.Activity;
import android.os.Bundle;
public class MainActivity extends Activity{
    @Override
    protected void onCreate(Bundle b) {
        super.onCreate(b);
        setContentView(R.layout.activity_main);
    }
}

```

Realizar los cambios correspondientes y ejecutar el proyecto para mostrar el resultado.

2. Obtener el texto del **EditText**.

Para obtener el valor de texto de un **EditText** se utiliza el método `getText()`. Este método regresa un objeto **Editable** no un **String**, pero si se utiliza `toString()` se obtiene el texto plano.

Por ejemplo, agregar un campo de texto y un botón, el cual al digitarse muestra el valor actual. Abrir el archivo **actividad_principal.xml** y el archivo **MainActivity.main** para modificar la plantilla. En la plantilla, colocar un **Button** debajo del **EditText**. Asignar un escucha al botón con el atributo `onClick`. El nombre del escucha es `verValor`.

```

<?xml version="1.0" encoding="utf-8"?>
<!-- Archivo: actividad_main.xml -->
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent" >
    <EditText
        android:id = "@+id/xet1"
        android:layout_width = "match_parent"
        android:layout_height = "wrap_content"
        android:layout_centerHorizontal = "true"
        android:layout_centerVertical = "true"
        android:hint = "Teléfono"
        android:inputType = "phone" />
    <Button
        android:id = "@+id/xbn1"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:layout_below = "@+id/xet1"
        android:layout_centerHorizontal = "true"
        android:onClick = "verValor"
        android:text = "Guardar" />
</RelativeLayout>

// Archivo: MainActivity.java
import android.app.Activity;
import android.os.Bundle;
public class MainActivity extends Activity {

```



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

Realizar los cambios correspondientes y ejecutar el proyecto para mostrar el resultado.

3. Tipos de Entrada en `EditText`.

El atributo `android:inputType` restringe el texto del usuario para ingresar caracteres válidos en el `EditText`. Este atributo determina el tipo de teclado virtual que aparecerá ante el usuario y otros tipos de comportamientos.

Por ejemplo, si el campo de texto valida un número telefónico, se utiliza el atributo `phone`.

```
<EditText
    android:id="@+id/xet1"
    android:layout_width="match_parent"
    android:inputType="phone"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:hint="Teléfono" />
```

Los valores típicos de `inputType`, son los siguientes:

Valor	Descripción
text	Recibe texto plano simple
textPersonName	Texto correspondiente al nombre de una persona
textPassword	Protege los caracteres que se van escribiendo con puntos
numberPassword	Contraseña de solo números enmascarada con puntos
textEmailAddress	Texto que será usado en un campo para emails
phone	Texto asociado a un número de teléfono
textPostalAddress	Para ingresar textos asociados a una dirección postal
textMultiLine	Permite múltiples líneas en el campo de texto
time	Texto para determinar la hora
date	Texto para determinar la fecha
number	Texto con caracteres numéricos
numberSigned	Permite números con signo
numberDecimal	Para ingresar números decimales

Tabla 1. Valores típicos de `inputType`.

Realizar los cambios correspondientes y ejecutar el proyecto para mostrar el resultado.

4. Limitación de la cantidad máxima de caracteres en la entrada.

Se puede limitar la cantidad máxima de caracteres que se reciben en el `EditText` con el atributo `android:maxLength`, el cual se especifica con un número entero positivo.

Por ejemplo, crear un `EditText` para el nombre del usuario que permita con `maxLength` el valor de 9 en el campo:

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/xet1"
```



```
android:inputType="text"
android:hint="Apodo"
android:maxLength="9"
android:layout_alignParentTop="true"
android:layout_centerHorizontal="true" />
```

Realizar los cambios correspondientes y ejecutar el proyecto para mostrar el resultado.

5. Un `EditText` de una sola línea.

Se puede reducir la capacidad de un campo de texto que muestre solamente una sola línea. Se utiliza el atributo `android:singleLine`; si no se especifica este atributo, se aceptarán varias líneas de texto y el teclado virtual usará el salto de línea como tecla de acción, en vez de la confirmación. El valor predeterminado es `false`, pero si se utiliza algún valor para `textInput`, ello implica que el valor `true` se asignará a `singleLine`, automáticamente.

Por ejemplo, agregar un campo de texto para el nombre del usuario. Al atributo `android:singleLine` se le asigna `true`. La tecla de confirmación cierra el teclado para terminar la edición.

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/xet1"
    android:hint="Nombre:"
    android:singleLine="true"
    android:layout_centerVertical="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />
```

Realizar los cambios correspondientes y ejecutar el proyecto para mostrar el resultado.

6. La propiedad `ems`.

La propiedad `em` determina el tamaño de un carácter con la cantidad de puntos de la fuente del texto. Es decir, `3em` representa 3 veces el tamaño de la fuente utilizada. Si la fuente mide 12 puntos, entonces `3em = 36puntos`. Se puede extender el ancho del `EditText` dependiendo del atributo `android:ems`. El atributo `android:width` debe tener asignado el valor `wrap_content`.

Por ejemplo, agregar un `EditText` con un ancho de `6em`. Se asigna el valor `6` al atributo `android:ems` y el tipo de entrada con texto plano.

```
<EditText
    android:id="@+id/xet1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:ems="6"
    android:hint="Un texto"
    android:inputType="text" />
```

Realizar los cambios correspondientes y ejecutar el proyecto para mostrar el resultado.

7. Ingreso de dígitos del 0 al 9.



Los caracteres se filtran con el atributo `android:digits` y una lista de elementos permitidos, por ejemplo, sólo a los dígitos del 0 al 9. Colocar los dígitos "01234567890 ". Se puede incluir un espacio vacío al final si también se desea ese carácter.

Por ejemplo, crear campo de texto que permita solamente números ingresados por el usuario. Se deben permitir solo 15 caracteres y sólo dígitos del 0 al 9. Incluir `android:digits` con la lista de valores indicada anteriormente y limitar la cantidad de caracteres a 15; además, utilizar el tipo de entrada `number` y una sola línea con `singleLine`.

```
<EditText
    android:id="@+id/xet1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:digits="1234567890 "
    android:singleLine="true"
    android:maxLength="15"
    android:inputType="number"
    android:ems="9"
    android:hint="RFC?" />
```

Realizar los cambios correspondientes y ejecutar el proyecto para mostrar el resultado.

8. Teclado virtual oculto desde Java.

El teclado se puede ocultar o cerrar manualmente, desde el código Java. Utilizar la clase `InputMethodManager` para la gestión de los métodos y procedimientos de entradas.

Por ejemplo, utilizar la promoción `InputMethodManager` del método `getSystemService()` que recibe la constante `INPUT_METHOD_SERVICE` y, por último, invocar al método `hideSoftInputFromInputMethod()`, el cual recibe una instancia de la entrada y es la interfaz de comunicación del view del teclado, con `getWindowsToken()`.

```
InputMethodManager imm =(InputMethodManager) getSystemService(INPUT_METHOD_SERVICE);
imm.hideSoftInputFromWindow(view.getWindowToken(), 0);
```

Realizar los cambios correspondientes y ejecutar el proyecto para mostrar el resultado.

9. Un EditText no editable.

Un `EditText` puede cambiar su estado habilitado a un estado deshabilitado con el atributo `android:enabled=false`.

Por ejemplo, también desde el archivo Java también se puede cambiar el estado utilizando el método `setEnabled(false)`.

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/xet1"
    android:enabled="false"
    android:hint="Edición deshabilitada" />
```

Realizar los cambios correspondientes y ejecutar el proyecto para mostrar el resultado.

10. Manejo del foco en EditText.

Cuando un `EditText` obtiene el foco se activa su cursor y el borde inferior cambia de color. Desde el archivo `MainActivity.java` se puede invocar al método `setFocusable(boolean)`, es decir:



```
findViewById(R.id.campo_sin_foco).setFocusable(false);
```

También, en el archivo `activity_main.xml`:

```
<EditText
    android:id="@+id/campo_sin_foco"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:focusable="false"
    android:hint="Sin foco" />
```

Realizar los cambios correspondientes y ejecutar el proyecto para mostrar el resultado.

11. Asignación del foco a un `EditText`.

Se puede asignar el foco a un `EditText` desde el archivo `MainActivity.java`, con el método `requestFocus()`. También se puede verificar si un componente posee el foco con el método `isFocusable()`.

Por ejemplo, colocar dos `EditText` en una plantilla:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <EditText
        android:id="@+id/xet1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="con foco al inicio" />
    <EditText
        android:id="@+id/xet2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="sin foco al inicio" />
</LinearLayout>
```

Ahora, para dirigir el foco al segundo `EditText` desde el archivo `MainActivity.java` utilizar el método `isFocusable()`. Entonces en el archivo Java se invoca también a `requestFocus()`.

```
import android.os.Bundle;
import android.app.Activity;
public class ActividadPrincipal extends Activity {
    @Override
    protected void onCreate(Bundle b) {
        super.onCreate(b);
        setContentView(R.layout.activity_main);
        if (findViewById(R.id.xet1).isFocusable()) {
            findViewById(R.id.xet2).requestFocus();
        }
    }
}
```

Realizar los cambios correspondientes y ejecutar el proyecto para mostrar el resultado.

12. Asignación del foco hacia otros componentes.



Es posible navegar entre diversos EditText para redirigir el foco según se necesite. Lo anterior depende del diseño de los componentes que se utilicen en la aplicación.

Atributo	Descripción	Método
android:nextFocusDown	Asigna el foco al siguiente campo de texto si el usuario navega hacia abajo.	setNextFocusDownId()
android:nextFocusLeft	Asigna el foco al siguiente campo de texto si el usuario navega hacia la izquierda.	setNextFocusLeftId()
android:nextFocusRight	Asigna el foco al siguiente campo de texto si el usuario navega hacia la derecha.	setNextFocusRightId()
android:nextFocusUp	Asigna el foco al siguiente campo de texto si el usuario navega hacia arriba.	setNextFocusUpId()

Tabla 2. Asignación del foco con los atributos de las etiquetas y su método invocador.

Por ejemplo, utilizar las teclas de flecha derecha e izquierda para redirigir el foco.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <EditText
        android:id="@+id/xet1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="busca el foco"
        android:nextFocusRight="@+id/xet2"
        android:singleLine="true" />
    <EditText
        android:id="@+id/xet2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Foco 2"
        android:nextFocusLeft="@+id/xet1"
        android:singleLine="true" />
</LinearLayout>
```

Realizar los cambios correspondientes y ejecutar el proyecto para mostrar el resultado.

13. Cambio de la posición del cursor.

El componente EditText posee el método setSelection() el cual permite mover el cursor del campo hacia una posición dentro del texto utilizando un índice.

Por ejemplo, en el archivo MainActivity.java asignar una posición para el cursor, utilizando el método setSelection(3), ya que la posición del campo inicia en el índice 0.

```
import android.app.*;
import android.os.Bundle;
import android.widget.*;
public class MainActivity extends Activity{
    @Override
    protected void onCreate(Bundle b) {
        super.onCreate(b);
        setContentView(R.layout.activity_main);
        EditText jet1 = (EditText) findViewById(R.id.xet1);
```



```

        jet1.setSelection(3);
    }
}

```

Enseguida, en el archivo `activity_main.xml`, crear una etiqueta `EditText` que posea el atributo `android:text="Cursor"`.

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <EditText
        android:id="@+id/xet1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="posicionando el cursor"
        android:hint="una posición"
        android:singleLine="true" />
</LinearLayout>

```

Realizar los cambios correspondientes y ejecutar el proyecto para mostrar el resultado.

14. Obtención de la posición del cursor.

Se puede obtener la posición del cursor con los métodos `getSelectionStart()`, el cual obtiene la posición izquierda y `getSelectionEnd()` el cual obtiene la posición final de la selección. Si no existe alguna selección, los dos métodos regresan la misma posición del cursor.

Por ejemplo, en el archivo `MainActivity.java`:

```

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.widget.EditText;
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle b) {
        super.onCreate(b);
        setContentView(R.layout.activity_main);
        EditText campol = (EditText) findViewById(R.id.xet1);
        Log.d("Obteniendo cursor...", String.valueOf(campol.getSelectionEnd()));
    }
}

```

Realizar los cambios correspondientes y ejecutar el proyecto para mostrar el resultado.

15. Seleccionando dinámicamente el texto.

En el archivo `MainActivity.java` se utiliza el método `setSelection(int ini, int fin)` con las posiciones inicial y final del texto. También se puede utilizar el método `extendSelection()` el cual toma el inicio desde la posición 0.

Por ejemplo, crear el `EditText` en el `activity_main.xml`:

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"

```




```

        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >
        <EditText
            android:id="@+id/xet1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="algún texto"
            android:hint="selección"
            android:singleLine="true" />
    </LinearLayout>

```

Enseguida, determinar cuál es el final de la primera palabra, si la posición inicial es 0 y la posición final es un espacio. Se recorre la cadena hasta el fin.

En el archivo **MainActivity.java**:

```

import android.app.*;
import android.os.*;
import android.text.Editable;
import android.widget.EditText;
public class MainActivity extends Activity {
    EditText jet1;
    Editable jet2;
    int ini = 0, fin = 0;
    @Override
    protected void onCreate(Bundle b) {
        super.onCreate(b);
        setContentView(R.layout.activity_main);
        jet1 = (EditText) findViewById(R.id.xet1);
        jet2 = jet1.getText();
        for (int i = ini; i < jet1.length(); i++)
            if (jet2.charAt(i) == ' ')
                fin = i;
        jet1.setSelection(ini, fin);
    }
}

```

Realizar los cambios correspondientes y ejecutar el proyecto para mostrar el resultado.

16. Selección completa del texto.

Con la invocación al método `selectAll()` se selecciona todo el texto.

Por ejemplo, el código en el **MainActivity.java**, se puede utilizar de la siguiente forma:

```

EditText jet1 = (EditText) findViewById(R.id.xet1);
jet1.selectAll();

```

17. Editor del método de entrada **InputMethodEntry(ime)**.

Al teclado virtual se le puede agregar una acción extra para una elección rápida si el usuario termina de ingresar el texto. Se utiliza el `android:imeOptions`. Lo anterior mostrará un icono equivalente a la acción indicada.

Constantes	Descripción	Icono
------------	-------------	-------









actionGo	Representa la acción de ejecutar alguna tarea que llevará al usuario a determinados resultados.	
actionSearch	Especifica que realizará una búsqueda con el texto que se acaba de agregar al campo de texto.	
actionSend	Se usa para indicar que se realizará una operación de envío de un contenido asociado al contenido del campo.	
actionNext	Tecla para realizar una operación del tipo "siguiente". Normalmente se usa para asignar el foco al TextField posterior.	
actionDone	Determina que se ha llevado a cabo satisfactoriamente la edición cerrando el teclado virtual.	
actionPrevious	Acción que lleva al usuario a un campo previamente aceptado.	

Tabla 3. Los valores típicos de android:imeOptions.

Por ejemplo, el uso de android:imeOptions="actionSend", en el archivo activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <EditText
        android:id="@+id/xet1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:imeOptions="actionSend"
        android:inputType="text"
        android:singleLine="true" />
</LinearLayout>
```

En los campos con varias líneas se muestra una tecla de salto. En la acción asignada a imeOptions se asigna el salto sin no se utiliza singleLine=true.

Por ejemplo, si se desea cambiar el texto del botón, se utiliza el atributo android:imeActionLabel. Por ejemplo, agregar la cadena o recurso del atributo:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <EditText
        android:id="@+id/xet1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```



```

        android:imeOptions="actionSend"
        android:imeActionLabel="Enviar"
        android:inputType="text"
        android:singleLine="true" />
</LinearLayout>

```

Realizar los cambios correspondientes y ejecutar el proyecto para mostrar el resultado. Documentar lo que sucede cuando el dispositivo está en posición horizontal.

18. Manejo de los eventos.

Para permitir la interacción del usuario con `EditText` y realizar otro tipo de operaciones, se utiliza la interface `TextWatcher` con el método `addTextChangedListener()`.

<code>afterTextChanged()</code>	Invocada cuando ya se realizó el cambio. Permite utilizar el texto actualizado.
<code>beforeTextChanged()</code>	Invocado antes de ingresar un texto. Permite conocer el estado y la sección por cambiar del texto.
<code>onTextChanged()</code>	Invocado cuando se ha reemplazado la sección del texto. Sus parámetros permiten conocer cual sección del texto anterior se cambió y los nuevos caracteres agregados.

Tabla 4. Los métodos de la clase `TextWatcher`.

Por ejemplo, abrir la plantilla `activity_main.xml` con varias líneas, por ejemplo3, utilizar los atributos `android:lines` y `android:maxLines`, como se indica enseguida:

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <EditText
        android:id="@+id/xet1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:layout_gravity="center_horizontal"
        android:inputType="textMultiLine"
        android:lines="3"
        android:maxLines="3" />
    <TextView
        android:id="@+id/xtv1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignEnd="@+id/xet1"
        android:layout_alignRight="@+id/xet1"
        android:layout_below="@+id/xet1"
        android:text="Small Text" />
</RelativeLayout>

```

En el archivo `MainActivity.java`, crear el objeto del `EditText` y asignar el método `addTextChangedListener()`. En la instancia del campo de texto `jtvt1`, se utiliza el método `afterTextChanged()` para conocer el tamaño y la posición inicial del texto en el `EditText`. En el interior del método `afterTextChanged()` en el `TextView` se incluye y se muestra el resultado de contar el número de caracteres del texto. Por ejemplo:

```

import android.app.Activity;
import android.os.Bundle;
import android.text.*;

```



```
import android.widget.*;
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle b) {
        super.onCreate(b);
        setContentView(R.layout.activity_main);
        EditText jet1 = (EditText) findViewById(R.id.xet1);
        jet1.addTextChangedListener(new TextWatcher() {
            @Override
            public void beforeTextChanged(CharSequence s, int start, int count, int after){}
            @Override
            public void onTextChanged(CharSequence s, int start, int before, int count) {}
            @Override
            public void afterTextChanged(Editable e) {
                TextView tv = (TextView) findViewById(R.id.xtv1);
                String t = String.valueOf(e.length());
                tv.setText(t);
            }
        });
    }
}
```

Realizar los cambios correspondientes y ejecutar el proyecto para mostrar el resultado.

19.Control de los eventos de los botones.

Para interactuar con los algunos botones, por ejemplo del teclado virtual, se utiliza el atributo `android:imeOptions`, la interface `TextView.OnEditorActionListener` y el método `onEditorAction()`, en el cual se incluyen las instrucciones de acción del botón; por ejemplo, digitar el botón de acción para mostrar un mensaje en la pantalla.

Por ejemplo, abrir el archivo `activity_main.xml` y realizar los cambios convenientes, es decir:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <EditText
        android:id="@+id/xet1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:ems="12"
        android:hint="búsqueda"
        android:imeOptions="actionSearch"
        android:singleLine="true" />
</RelativeLayout>
```

Abrir el archivo `MainActivity.java` para instanciar el `EditText`. Asignar los métodos `setOnEditorActionListener()` y `onEditorAction()`. Con el componente de la clase `Toast` se muestra el resultado en un mensaje momentáneo. Para comparar el identificador se utiliza la constante `IME_ACTION_SEARCH` de `EditorInfo`, el cual utiliza sus parámetros para obtener el resultado deseado, en este caso su parámetro `v`.

```
import android.app.Activity;
import android.os.Bundle;
import android.view.KeyEvent;
```



```

import android.view.inputmethod.*;
import android.widget.*;
public class MainActivity extends Activity {
    EditText jet1;
    @Override
    protected void onCreate(Bundle b) {
        super.onCreate(b);
        setContentView(R.layout.activity_main);
        jet1 = (EditText) findViewById(R.id.xet1);
        jet1.setOnEditorActionListener(new TextView.OnEditorActionListener() {
            @Override
            public boolean onEditorAction(TextView v, int id, KeyEvent ke) {
                boolean b = false;
                if(id == EditorInfo.IME_ACTION_SEARCH){
                    Toast.makeText(MainActivity.this, "búsqueda..." +
v.getText().toString(), Toast.LENGTH_LONG).show();
                    InputMethodManager imm = (InputMethodManager)
getSystemService(INPUT_METHOD_SERVICE);
                    imm.hideSoftInputFromWindow(v.getWindowToken(), 0);
                    b = true;
                }
                return b;
            }
        });
    }
}

```

Realizar los cambios correspondientes y ejecutar el proyecto para mostrar el resultado.

20. Uso de `OnFocusChangeListener` para cambiar el foco.

Utilizar el método `setOnFocusChangeListener()` y `onFocusChange()` para utilizar una instancia del escucha a los campos de texto. Por ejemplo, cambiar el atributo de una imagen a través de un `EditText` el cual posee el valor del cambio. Utilizar `RelativeLayout` y `EditText` para números decimales `numberDecimal`. Incluir un `ImageView` que limite con el campo de texto. Utilizar de fondo (con `android:src`) una imagen de color gris y tamaño de 24x24dp. Es decir:

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:focusableInTouchMode="true"
    android:orientation="vertical" >
    <EditText
        android:id="@+id/xet1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
        android:layout_gravity="center_horizontal"
        android:layout_toEndOf="@+id/xiv1"
        android:layout_toRightOf="@+id/xiv1"
        android:inputType="numberDecimal" />
    <ImageView
        android:id="@+id/xiv1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"

```



```

        android:layout_alignParentStart="true"
        android:layout_centerVertical="true"
        android:layout_marginEnd="24dp"
        android:layout_marginRight="24dp" />
</RelativeLayout>

```

En el archivo `MainActivity.java`, se utiliza `android:focusableInTouchMode` en `RelativeLayout` para asignarle el foco cuando el usuario ingrese, por lo que `EditText` es el que posee el foco al inicio. Agregar `OnFocusChangeListener` a `jet1` utilizando `setOnFocusChangeListener()`, por ejemplo:

```

EditText campoDescuento = (EditText) findViewById(R.id.campo_descuento);
campoBusqueda.setOnFocusChangeListener(new View.OnFocusChangeListener() {
    @Override
    public void onFocusChange(View v, boolean hasFocus) {
        :
    }
});

```

Donde, `v` es alterado por el foco y `hasFocus` verifica si posee el foco para cambiar la imagen y su atributo. Las nuevas clases son `DrawableCompat` (y los métodos `getDrawable()`, `wrap()` y `setTint()`) y `ContextCompat` (con el método `getColor()`). El archivo `MainActivity.java` modificado es el siguiente:

```

import android.app.Activity;
import android.graphics.drawable.Drawable;
import android.os.Bundle;
import android.view.View;
import android.widget.*;
import androidx.core.content.ContextCompat;
import androidx.core.graphics.drawable.DrawableCompat;
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle b) {
        super.onCreate(b);
        setContentView(R.layout.activity_main);
        EditText jet1 = (EditText) findViewById(R.id.xet1);
        jet1.setOnFocusChangeListener(new View.OnFocusChangeListener() {
            @Override
            public void onFocusChange(View v, boolean hasFocus) {
                if (hasFocus) {
                    ImageView iconoDescuento = (ImageView) findViewById(R.id.xiv1);
                    Drawable d = iconoDescuento.getDrawable();
                    d = DrawableCompat.wrap(d);
                    DrawableCompat.setTint(d,
                        ContextCompat.getColor(MainActivity.this, R.color.micolor));
                } // micolor es un archivo xml en la carpeta color.xml.
            }
        });
    }
}

```

Realizar los cambios correspondientes y ejecutar el proyecto para mostrar el resultado.

21. Cambio de color de hint.

El atributo `android:textColorHint` permite cambiar el color de hint del `EditText`. El color se puede especificar con un valor textual o numérico (por ejemplo, octal o hexadecimal) directamente asignado en el atributo de la etiqueta o con una etiqueta `<color></color>` en el archivo `colors.xml`.



Por ejemplo:

```
<EditText
    android:id="@+id/xet1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="20dp"
    android:hint="un color diferente"
    android:singleLine="true"
    android:textColorHint="@color/colorHint" />
```

Realizar los cambios correspondientes y ejecutar el proyecto para mostrar el resultado.

22. Cambio del color del texto seleccionado.

Si se desea cambiar el color del texto seleccionado, se utiliza el atributo `android:textColorHighLight`, asignando un color en la forma como se mencionó en el ejercicio anterior.

```
<EditText
    android:id="@+id/xet1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="20dp"
    android:hint="un color diferente"
    android:singleLine="true"
    android:textColorHighlight="#123ABC" />
```

Realizar los cambios correspondientes y ejecutar el proyecto para mostrar el resultado.

23. Centrado del texto.

Para centrar el texto en forma horizontal o vertical se puede utilizar el atributo `android:gravity` o `android:textAlignment` asignando el valor `center`:

```
<EditText
    android:id="@+id/xet1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="20dp"
    android:hint="Texto centrado"
    android:singleLine="true"
    android:gravity="center" />
```

Realizar los cambios correspondientes y ejecutar el proyecto para mostrar el resultado.

24. Cambio de color del borde.

Abrir el archivo `styles.xml` para crear un nuevo estilo que herede de `ThemeAppCompatLight` y las propiedades `colorControlNormal` y `colorControlActivated`, como se indica en el siguiente ejemplo:

```
<style name="miEstilo" parent="Theme.AppCompat.Light">
    <item name="colorControlNormal">#CD82C7</item>
    <item name="colorControlActivated">#123ABC</item>
    <item name="android:textColorHighLight">#D0ADD6</item>
</style>
```

En el archivo `activity_main.xml` se asigna el tema `miEstilo`:

```
<EditText
    android:id="@+id/xet1"
```



```

android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_marginBottom="20dp"
android:hint="nuevo color del borde"
android:inputType="textNoSuggestions"
android:singleLine="true"
android:theme="@style/miEstilo" />

```

Realizar los cambios correspondientes y ejecutar el proyecto para mostrar el resultado.

25. Cambio del tamaño de la fuente.

Utilizar el atributo `android:textSize` y el prefijo `sp` o `scaleable pixels`, como se indica en el siguiente ejemplo:

```

<EditText
    android:id="@+id/xet1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="nuevo tamaño de fuente"
    android:singleLine="true"
    android:textSize="34sp" />

```

Realizar los cambios correspondientes y ejecutar el proyecto para mostrar el resultado.

26. Uso de AutoCompleteTextView.

Un equivalente de `EditText` es `AutoCompleteTextView` el cual permite mostrar sugerencias textuales que coincidan con algunos caracteres iniciales que el usuario digita. El usuario puede seleccionar alguna de las palabras sugeridas para reemplazarla y actualizarla en campo de texto `AutoCompleteTextView`.

En el archivo `activity_main.xml`, insertar la nueva etiqueta que representa el `AutoCompleteTextView` y sus atributos básicos. La lista desplegable de las palabras puede provenir de una base de datos o de otra fuente textual. En el archivo `MainActivity.java` instanciar y convertir la referencia a este nuevo campo de edición, con `AutoCompleteTextViewjactv1 = (AutoCompleteTextView) findViewById(R.id.xactv1);`,

Algunos métodos de `AutoCompleteTextView` son:

<code>getAdapter()</code>	Devuelve un adaptador de lista filtrable utilizado para la finalización automática.
<code>getCompletionHint()</code>	Devuelve el texto de sugerencia opcional que se muestra al final de la lista coincidente.
<code>getDropDownAnchor()</code>	Devuelve el identificador de la vista a la que se adjunta la lista desplegable de autocompletar.
<code>getListSelection()</code>	Devuelve la posición de la selección de vista desplegable, si hay alguna.
<code>isPopupShowing()</code>	Indica si se muestra el menú emergente.
<code>setText(CharSequence text, boolean filter)</code>	Establece el texto, pero puede deshabilitar el filtrado.
<code>showDropDown()</code>	Muestra el menú desplegable en la pantalla.

Tabla 5. Métodos de la clase `AutoCompleteTextView`.

Por ejemplo, realizar las siguientes modificaciones, como se indica en el siguiente ejemplo:

```

<AutoCompleteTextView
    android:id="@+id/xactv1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="autocompletado de palabras "

```




```
android:completionThreshold="1"  
android:singleLine="true" />
```

En el archivo **MainActivity.java**, se incluye la lista de palabras alternativas y una clase adaptadora para incluya la lista sugerida en un arreglo de cadenas:

```
import android.os.Bundle;  
import android.widget.*;  
import androidx.appcompat.app.AppCompatActivity;  
public class MainActivity extends AppCompatActivity {  
    public static String[] MI_LISTA = {"Android", "Arriba", "África", "Asia", "América",  
    "A", "AMOR"};  
    @Override  
    protected void onCreate(Bundle b) {  
        super.onCreate(b);  
        setContentView(R.layout.activity_main);  
        AutoCompleteTextView jactv1 = (AutoCompleteTextView) findViewById(R.id.xactv1);  
        ArrayAdapter<String> aa = new ArrayAdapter<>(this,  
        android.R.layout.simple_dropdown_item_1line, MI_LISTA);  
        jactv1.setAdapter(aa);  
    }  
}
```

NOTA. Realizar los cambios correspondientes en cada ejercicio y ejecutarlo para mostrar el resultado. Entregar un reporte que contenga únicamente las pantallas obtenidas de cada uno de los 26 ejercicios desarrollados. Enviar el reporte, con la sintaxis **AlumnoEditTextGrupo.doc**, al sitio indicado por el profesor.