



**Asignatura:** Application Development for Mobile Devices.

**Tarea 23:** Drag and Drop.

Android permite que el usuario arrastre o suelte los datos desde un `View` a otro `View`, en la plantilla actual, utilizando un gesto gráfico de arrastrar o soltar. Se tienen los siguientes tres componentes importantes que permiten la funcionalidad **Drag & Drop**, de arrastrar y soltar:

- La clase del evento de arrastre.
- Escuchas de arrastre.
- Métodos y clases de ayuda.

### El Proceso Drag & Drop

Básicamente, se tienen cuatro pasos o estados durante el proceso de arrastrar y soltar:

- **Iniciado:** Este evento se produce cuando se empieza a arrastrar un elemento en una plantilla, invocando al método `startDrag()` para indicar al sistema el inicio del arrastre.

Los argumentos del método `startDrag()` indican los datos por ser arrastrados, los metadatos para estos datos, y una invocación para dibujar la sombra del arrastre.

El sistema responde primero indicándole a la aplicación que obtenga la sombra de arrastre.

A continuación, muestra la sombra de arrastre en el dispositivo.

Enseguida, el sistema envía un evento de arrastre con el tipo de acción `ACTION_DRAG_STARTED` a los escuchas de eventos de arrastre registrados para todos los objetos `View` en la plantilla actual.

Para continuar recibiendo los eventos de arrastre, incluyendo un posible evento de soltar, un escucha de eventos de arrastrar debe devolver `true`.

Si el escucha de eventos de arrastre devuelve `false`, entonces no recibirá los eventos de arrastre de la operación actual hasta que el sistema envíe un evento de arrastre con el tipo de acción `ACTION_DRAG_ENDED`.

- **Continuando:** El usuario continúa el arrastre. El sistema envía una acción `ACTION_DRAG_ENTERED` seguida de la acción `ACTION_DRAG_LOCATION` al escucha de eventos de arrastre registrado para el `View` donde se ingresa el punto de arrastre. El escucha puede optar por alterar la apariencia del objeto `View` en respuesta al evento o puede reaccionar resaltando el `View`.

El escucha del evento de arrastre recibe una acción `ACTION_DRAG_EXITED` después de que el usuario ha movido la sombra de arrastre fuera del cuadro delimitador del `View`.

- **Soltado:** El usuario suelta al elemento arrastrado dentro del cuadro delimitador del `View`. El sistema envía al escucha del objeto `View` un evento de arrastre con el tipo de acción `ACTION_DROP`.

- **Finalizado:** Justo después del tipo de acción `ACTION_DROP`, el sistema envía un evento de arrastre con el tipo de acción `ACTION_DRAG_ENDED` para indicar que la operación de arrastre ha terminado.

### La clase `DragEvent`

El `DragEvent` representa un evento que envía el sistema en varias ocasiones durante una operación de arrastrar y soltar.

Esta clase proporciona algunos métodos y constantes importantes que se utilizan durante el proceso de arrastrar o soltar.



## CONSTANTES

Los siguientes son todos los enteros constantes disponibles que son parte de la clase `DragEvent`.

Paso	Constantes y su Descripción
1	<code>ACTION_DRAG_STARTED</code> Indica el inicio de la operación Drag & Drop.
2	<code>ACTION_DRAG_ENTERED</code> Le indica a un <code>View</code> que el punto de arrastre ha ingresado al delimitador del <code>View</code> .
3	<code>ACTION_DRAG_LOCATION</code> Enviado a un <code>View</code> después del <code>ACTION_DRAG_ENTERED</code> si la sombra de arrastre se encuentra todavía dentro del limitador del objeto <code>View</code> .
4	<code>ACTION_DRAG_EXITED</code> Indica que el usuario ha movido la sombra de arrastre fuera del delimitador del <code>View</code> .
5	<code>ACTION_DROP</code> Le indica a un <code>View</code> que el usuario ha liberado la sombra de arrastre y que el punto de arrastre se encuentra dentro del delimitador del <code>View</code> .
6	<code>ACTION_DRAG_ENDED</code> Le indica a un <code>View</code> que la operación de Drag & Drop ha concluido.

## MÉTODOS

A continuación se presentan algunos de los métodos más importantes utilizados de la clase `DragEvent`.

Paso	Constantes y su Descripción
1	<code>int getAction()</code> Verifica el valor de la acción de este evento.
2	<code>ClipData getClipData()</code> Regresa el objeto <code>ClipData</code> enviado al sistema como parte de la llamada a <code>startDrag()</code> .
3	<code>ClipDescription getClipDescription()</code> Regresa el objeto <code>ClipDescription</code> contenido en el <code>ClipData</code> .
4	<code>boolean getResult()</code> Regresa una indicación del resultado de la operación Drag & Drop.
5	<code>float getX()</code> Obtiene la coordenada X del punto de arrastre.
6	<code>float getY()</code> Obtiene la coordenada Y del punto de arrastre.
7	<code>String toString()</code> Regresa la representación de una cadena de este objeto <code>DragEvent</code> .

**Escuchando el evento de arrastre.**



Si se desea que cualquiera de los `View` dentro de una plantilla deba responder a un evento de arrastre, entonces el `View` debe implantar el método `View.OnDragListener` o configurar al método `onDragEvent(DragEvent)`. Cuando el sistema llama al método o al escucha, les pasa un objeto `DragEvent`, como el mencionado anteriormente. Se puede tener un escucha y un método para el objeto `View`. Si esto sucede, el sistema llama primero al escucha y luego define la invocación del método, siempre y cuando el escucha devuelva `true`.

La combinación del método `onDragEvent(DragEvent)` y `View.OnDragListener` es análoga a la combinación de la `onTouchEvent()` y `View.OnTouchListener` utilizados con eventos táctiles en versiones anteriores de Android.

### Inicio de un evento de arrastre

Se comienza con la creación de un `ClipData` y `ClipData.Item` para los datos se desea mover. Como parte del objeto `ClipData`, se suministran los metadatos que se almacenan en un objeto `ClipDescription` dentro del `ClipData`. Para una operación de arrastrar y soltar que no represente el movimiento de datos, es posible que desee utilizar `null` en lugar de un objeto real.

Enseguida, se puede heredar de `View.DragShadowBuilder` para crear una sombra de arrastre para arrastrar el `View` o simplemente utilizar `View.DragShadowBuilder(View)` para crear una sombra de arrastre predeterminada que es del mismo tamaño que el argumento `View` que se le pasó, con el punto digitado centrado en la sombra de arrastre.

### Ejemplo

El ejemplo siguiente muestra la funcionalidad del Drag & Drop utilizando un escucha de evento `View.setOnLongClickListener()` junto con `View.OnDragEventListener()`.

	Descripción
1	Crear una aplicación Android y nombrarla <code>DragNDropDemo</code> bajo el paquete <code>com.example.dragndropdemo</code> . Al crear el proyecto, configurar el <b>Target SDK</b> y compilar con la version más reciente del Android SDK con las APIs de más alto nivel.
2	Modificar el archive <code>src/MainActivity.java</code> y agregar el código para definir los escuchas de eventos y la invocación a los métodos de la imagen.
3	Copiar la imagen <code>logo.png</code> en la carpeta <code>res/drawable</code> . Se pueden utilizar imágenes con diferente resolución.
4	Modificar el archive XML de la plantilla <code>res/layout/activity_main.xml</code> para definir la vista predeterminada de la imágenes.
5	Ejecutar la aplicación para lanzar el emulador y verificar los cambios realizados a la aplicación.

A continuación se muestra el contenido del archivo `MainActivity` modificado `src/com.example.dragndropdemo/MainActivity.java`. Este archivo puede incluir cada uno de los métodos fundamentales del ciclo de vida

```
package com.example.dragndropdemo;
import android.os.Bundle;
import android.app.Activity;
import android.content.ClipData;
import android.content.ClipDescription;
import android.util.Log;
import android.view.DragEvent;
```



```
import android.view.View;
import android.view.View.DragShadowBuilder;
import android.view.View.OnDragListener;
import android.widget.*;
public class MainActivity extends Activity{
    ImageView ima;
    private static final String IMAGEVIEW_TAG ="Android Logo";
    String msg;
    private android.widget.RelativeLayout.LayoutParams layoutParams;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ima =(ImageView) findViewById(R.id.iv_logo);
        // Sets the tag
        ima.setTag(IMAGEVIEW_TAG);
        ima.setOnLongClickListener(new View.OnLongClickListener(){
            @Override
            public boolean onLongClick(View v){
                ClipData.Item item = new ClipData.Item((CharSequence)v.getTag());
                String[] mimeTypes = {
                    ClipDescription.MIMETYPE_TEXT_PLAIN};
                ClipData dragData = new ClipData(v.getTag().toString(), mimeTypes,
item);

                // Instantiates the drag shadow builder.
                View.DragShadowBuilder myShadow = new DragShadowBuilder(ima);
                // Starts the drag
                v.startDrag(
                    dragData,// the data to be dragged
                    myShadow,// the drag shadow builder
                    null,// no need to use local data
                    0// flags (not currently used, set to 0)
                );
                return true;
            }
        });
        // Create and set the drag event listener for the View
        ima.setOnDragListener(new OnDragListener(){
            @Override
            public boolean onDrag(View v,DragEvent event){
                switch(event.getAction()){
                    case DragEvent.ACTION_DRAG_STARTED:
                        layoutParams =(RelativeLayout.LayoutParams) v.getLayoutParams();
                        Log.d(msg,"Action is DragEvent.ACTION_DRAG_STARTED");
                        // Do nothing
                        break;
                    case DragEvent.ACTION_DRAG_ENTERED:
                        Log.d(msg,"Action is DragEvent.ACTION_DRAG_ENTERED");
                        int x_cord =(int)event.getX();
                        int y_cord =(int)event.getY();
                        break;
                    case DragEvent.ACTION_DRAG_EXITED :
                        Log.d(msg,"Action is DragEvent.ACTION_DRAG_EXITED");
                        x_cord =(int)event.getX();
                        y_cord =(int)event.getY();
                        layoutParams.leftMargin = x_cord;
                        layoutParams.topMargin = y_cord;
                        v.setLayoutParams(layoutParams);
                }
            }
        });
    }
}
```



```

        break;
    case DragEvent.ACTION_DRAG_LOCATION :
        Log.d(msg, "Action is DragEvent.ACTION_DRAG_LOCATION");
        x_cord = (int)event.getX();
        y_cord = (int)event.getY();
        break;
    case DragEvent.ACTION_DRAG_ENDED :
        Log.d(msg, "Action is DragEvent.ACTION_DRAG_ENDED");
        // Do nothing
        break;
    case DragEvent.ACTION_DROP:
        Log.d(msg, "ACTION_DROP event");
        // Do nothing
        break;
    default:break;
}
return true;
}
});
}
}

```

Enseguida el contenido del archivo `res/layout/activity_main.xml`:

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/container"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <ImageView
        android:id="@+id/iv_logo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/logo"
        android:contentDescription="@string/drag_drop" />
</RelativeLayout>

```

Enseguida el contenido del archivo `res/values/strings.xml` para definir dos nuevas constantes:

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <stringname="app_name">DragNDropDemo</string>
    <stringname="action_settings">Settings</string>
    <stringname="hello_world">Drag & Drop</string>
    <stringname="drag_drop">Click en la imagen para arrastrala</string>
</resources>

```

A continuación se presenta el contenido predeterminado de `AndroidManifest.xml`:


```

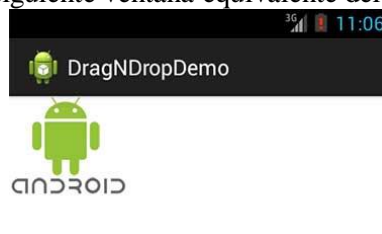
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package = "com.example.guidemo"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk
        android:minSdkVersion="16"
        android:targetSdkVersion="17"/>
    <application
        android:allowBackup="true"

```



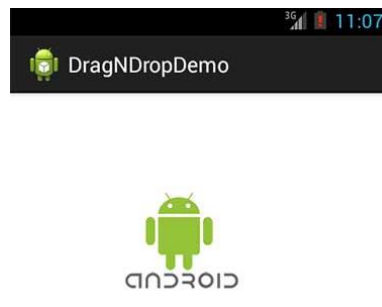
```
android:icon="@drawable/ic_launcher"
android:label="@string/app_name"
android:theme="@style/AppTheme" >
<activity
    android:name="com.example.guidemo.MainActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <actionandroid:name="android.intent.action.MAIN"/>
        <categoryandroid:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
</activity>
</application>
</manifest>
```

Ejecutar la aplicación DragNDropDemo. Se supone que se ha creado el AVD mientras se hizo la configuración del entorno. Para ejecutar la aplicación, digitar en el icono Run  de la barra de herramientas. Si todo está bien con la configuración de la aplicación, se mostrará siguiente ventana equivalente del emulador:



**Figura 1.** El objeto en su posición inicial.

Ahora, digitando un largo clic en el logo de Android se verá que la imagen del logotipo se mueve un poco de su lugar, después de 1 segundo es el momento en que se debe comenzar a arrastrar la imagen. Se puede arrastrar por la pantalla y soltarla en una nueva ubicación.



**Figura 2.** El objeto en su nueva posición.

**NOTA:** Capturar las imágenes de la ejecución del ejercicio, en un documento y guardarlo con la sintaxis AlumnoTarea23Grupo.pdf. Enviar el archivo al sitio indicado por el profesor.