Tema 6. Transiciones. Palette

Tema 6. Transiciones. Palette

ÍNDICE.

1.1 Palette

Librería Palette

Hasta ahora, hemos definido los colores básicos de nuestra aplicación (*primary color, dark primary color y accent color*) en el momento del diseño de la misma. Sin embargo, nos puede interesar modificar los colores principales de nuestra interfaz en tiempo de ejecución, y en función de los colores de las imágenes que vayamos cargando a cada momento.

Para realizar esta tarea Android proporciona una librería conocida como **Palette**, cuyo principal objetivo es extraer de una imagen los colores principales que la componen para poder así utilizarlos en nuestra interfaz.

Con la utilización de Palette, podemos conseguir 6 colores principales (3 intensos "vibrant" y 3 tenues "muted") que son:

- Vibrant
- Vibrant Dark
- Vibrant Light
- Muted
- Muted Dark
- Muted Light

Además, tendremos la posibilidad de conseguir los colores apropiados para utilizar en textos sobre fondos del color seleccionado.

Para poder usar la librería Palette, tendremos que enlazarla en el fichero *build.gradle:*

```
dependencies {

    implementation 'androidx.palette:palette:1.0.0'
}
```

Una vez hecho esto, podemos utilizar la librería llamándola de forma síncrona o asíncrona. Por simplicidad, vamos a ver el ejemplo haciendo la llamada de forma **asíncrona**:

En Java:

```
Bitmap bitmap = BitmapFactory.decodeResource(getResources(),
R.drawable.image);

Palette.from(bitmap).generate(new Palette.PaletteAsyncListener() {
     public void onGenerated(Palette p) {
       Palette.Swatch vibrant = palette.getVibrantSwatch();
                                  Palette.Swatch darkvibrant =
palette.getDarkVibrantSwatch();
                                 Palette.Swatch lightvibrant =
palette.getLightVibrantSwatch();
       Palette.Swatch muted = palette.getMutedSwatch();
       Palette.Swatch darkmuted = palette.getDarkMutedSwatch();
                                   Palette.Swatch lightmuted =
palette.getLightMutedSwatch();
     }
}
```

En Kotlin:

```
val bitmap:Bitmap = BitmapFactory.decodeResource(getResources(), R.

Palette.from(bitmap).generate {

 val vibrant: Palette.Swatch? = it?.vibrantSwatch
 val darkvibrant: Palette.Swatch? = it?.darkVibrantSwatch
 val lightvibrant: Palette.Swatch? = it?.lightVibrantSwatch
 val muted: Palette.Swatch? = it?.mutedSwatch
 val darkmuted: Palette.Swatch? = it?.darkMutedSwatch
 val lightmuted: Palette.Swatch? = it?.lightMutedSwatch
```

```
})
```

Generamos la paleta de colores que componen la imagen R.drawable.image, utilizando las estructuras propias de la paleta, que son los llamados Swatch.

Palette intentará generar una estructura swatch por cada uno de los colores principales, haciendo una llamada al método *getXXXSwatch()*, donde *XXX* representa cada uno de los colores principales. La estructura swatch nos dará, además del color, información adicional que puede sernos de utilidad, por ejemplo: el color del texto en general y el color del título del texto a aplicar sobre el color de fondo seleccionado, consiguiendo así una gama de colores perfectamente acorde.

Cuando vayamos a utilizar los colores generados, hay que tener en cuenta que Palette no siempre podrá generar todas las tonalidades, por lo que deberemos comprobar siempre que el color generado es distinto de *null.* El color del swatch lo obtendremos mediante el método *getRgb()*, el color del texto principal se obtendrá mediante *getBodyTextColor()* y el color de los títulos mediante *getTitleTextColor()*.
En Java:

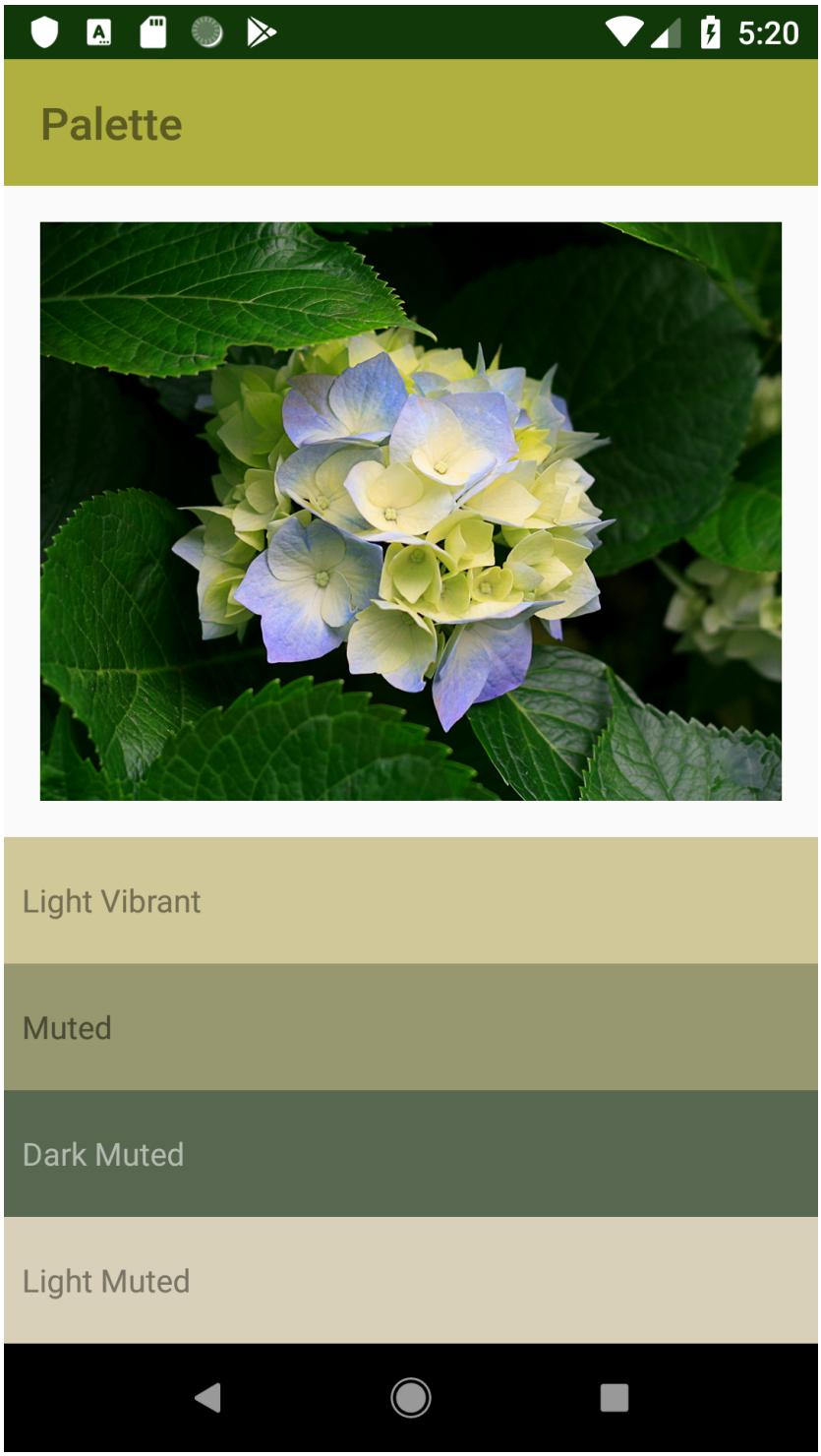
```
if (vibrant != null) {

        int backgroundColor = vibrant.getRgb();
        int titleColor = vibrant.getTitleTextColor();
 }
```

En Kotlin, aplicamos el color *vibrant* sobre el fondo de la ToolBar y sobre el título de la aplicación:

```
if (vibrant != null) {
    toolbar.setBackgroundColor(vibrant.rgb)
    toolbar.setTitleTextColor(vibrant.titleTextColor)
}
```

La gama de 6 colores obtenida para esta imagen sería la indicada, teniendo en cuenta que se ha aplicado el color vibrant sobre la Toolbar y el vibrant dark sobre la Statusbar:



1.2 Transition Manager

Transition Manager

Framework utilizado para animar automáticamente las vistas de un layout. Podemos especificar el tipo de transición (animación) a aplicar sobre una escena usando el método **beginDelayedTransition**.

Este método presenta la siguiente sintaxis:

```
static void beginDelayedTransition(ViewGroup root, Transition transition);
```

- root: escena sobre la que se ejecuta la transición.
- transition: tipo de transición. Si no se especifica ningún tipo, TransitionManager aplica la transición por defecto (AutoTransition).

Tipos de Transiciones

- *AutoTransition*. Se aplica por defecto cuando no se especifica ninguna transición en el segundo argumento de beginDelayedTransition.
- *Slide.* Deslizamiento de la vista hacia uno de los lados (arriba, abajo, derecha o izquierda).
- *Explode.* Deslizamiento de la vista hacia una dirección calculada. Por defecto la propagación de la transición es circular.
- *Fade*. Aparición (fade in) y desaparición (fade out) de la vista.
- *ChangeBounds*. Anima cambios de tamaño y posición del componente.
- *ChangeImageTransform.* Anima cambios en la propiedad ScaleType de un ImageView. Se

suele utilizar conjuntamente con ChangeBounds.

- *TransitionSet.* Conjunto de transiciones. Pueden empezar en paralelo o de manera secuencial, según se indique en el método setOrdering().

Las transiciones se pueden implementar mediante código xml o java. En xml, tendremos que almacenarlas en la carpeta *res/transition.*

Ejemplo. Transición slide con xml

Partiendo del siguiente layout o escena (*activity_main.xml*):

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

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="300dp"
        android:id="@+id/image"
        android:src="@drawable/dock"
        android:scaleType="centerCrop"
        android:layout_gravity="center"/>

    <com.google.android.material.floatingactionbutton.FloatingActio
        android:id="@+id/fab"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:elevation="6dp"
        app:fabSize="normal"
        android:layout_margin="16dp"
        android:layout_gravity="center_horizontal"
        android:src="@mipmap/ic_tick" />
```

```
</LinearLayout>
```

Definiríamos el fichero *res/transition/*slide*.xml* como se indica:

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

En la actividad principal *MainActivity.java* añadiríamos:

```
final FloatingActionButton fab = (FloatingActionButton) findViewByI
final ImageView img = (ImageView) findViewById(R.id.image);

fab.setOnClickListener(new View.OnClickListener() {
 @Override
 public void onClick(View v) {
     //Inflamos la transición, en este caso de tipo Slide
     Slide slide = (Slide) TransitionInflater.from(MainActivity.thi
     //Realizamos un deslizamiento del componente hacia arriba
     slide.setSlideEdge(Gravity.TOP);
     ViewGroup root = (ViewGroup) findViewById(R.id.content);
     TransitionManager.beginDelayedTransition(root, slide);
     if (img.getVisibility() == View.VISIBLE)
       img.setVisibility(View.INVISIBLE);
     else
       img.setVisibility(View.VISIBLE);
 }
});
```

En Kotlin:

```
fab.setOnClickListener { view ->
val slide = TransitionInflater.from(this@MainActivity).inflateTrans
 slide slideEdge = Gravity.TOP
```

```
 val root = findViewById<View>(R.id.content) as ViewGroup
 TransitionManager.beginDelayedTransition(root, slide)
 if (img.visibility == View.VISIBLE)
 img.visibility = View.INVISIBLE
else
img.visibility= View.VISIBLE
}
```

Ejemplo. Transición slide con java

Tomando como escena el mismo layout (*activity_main.xml*) que hemos utilizado en el ejemplo anterior, en la actividad principal *MainActivity.java* tendríamos que añadir:

```
final FloatingActionButton fab = (FloatingActionButton) findViewByI
final ImageView img = (ImageView) findViewById(R.id.image);

fab.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Slide slide = new Slide();
        slide.setSlideEdge(Gravity.TOP);
        ViewGroup root = (ViewGroup) findViewById(R.id.content);
        TransitionManager.beginDelayedTransition(root, slide);
        if (img.getVisibility() == View.VISIBLE)
            img.setVisibility(View.INVISIBLE);
        else
            img.setVisibility(View.VISIBLE);
    }
});
```

En Kotlin:

```
fab.setOnClickListener { view ->
val slide = Slide()
 slide.slideEdge = Gravity.TOP
```

```
 val root = findViewById<View>(R.id.content) as ViewGroup
 TransitionManager.beginDelayedTransition(root, slide)
 if (img.visibility == View.VISIBLE)
 img.visibility = View.INVISIBLE
else
img.visibility= View.VISIBLE
}
```

1.3 Transiciones entre actividades

Transición animada entre actividades

Consiste en realizar una transición animada entre dos actividades. La animación a realizar estará predefinida por Android.

Para realizar la transición entre dos actividades es necesario:

1. Permitir transiciones en el tema de la aplicación.
2. Especificar las transiciones de las actividades: entrada y salida.
3. Definir las transiciones en ficheros .xml, dentro de res/transition.
4. Usar el método ActivityOptionsCompat.makeSceneTransitionAnimation() para realizar la transición entre actividades.

Veamos los pasos a seguir con un ejemplo:

Ejemplo. Transición animada entre actividades.

En la primera actividad de nuestro ejemplo, tendremos un ListView donde, en cada ítem aparecerá un texto (nombre del usuario) y una imagen, y en la segunda actividad aparecerá desplegada la misma imagen que teníamos en la primera y su texto correspondiente. Al pulsar sobre un ítem del ListView de la primera escena o actividad, se aplicará una animación que nos llevará hasta la segunda actividad.

- Permitimos las transiciones en el tema, y especificamos la transición animada de entrada y salida entre actividades en el fichero de estilos (*styles.xml*):

```
<resources>
 <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
```

```
    <!-- enable window content transitions -->
    <item name="android:windowContentTransitions">true</item>

    <!-- specify transitions -->
    <item name="android:windowEnterTransition">@transition/explode</
    <item name="android:windowExitTransition">@transition/explode</i
  </style>
</resources>
```

- Definimos las transiciones en *res/transition.* El fichero *explode.xml* se definiría como sigue:

```
<?xml version="1.0" encoding="utf-8"?>

<explode xmlns:android="http://schemas.android.com/apk/res/android"
</explode>
```

- En la primera actividad definimos la transición animada, al hacer click sobre un ítem del ListView (lstOpciones).

```
lstOpciones.setOnItemClickListener(new AdapterView.OnItemClickListe
    public void onItemClick(AdapterView<?> a, View v, int position,

        Intent intent = new Intent(this, MainActivity2.class);

        //Animación aplicada sobre ambas actividades
        Bundle options = ActivityOptionsCompat.makeSceneTransitionA


        AndroidUsers item =((AndroidUsers)a.getItemAtPosition(posit
        options.putString("NOM",item.getTitulo());
        options.putInt("LOGO", item.getLogo());

        //Añadimos la información al Intent
        intent.putExtras(options);
        startActivity(intent, options);
```

```
 }
});
```

En Kotlin, sobre un RecyclerView:

```
adaptador.onClick = {
   val intent = Intent(this, MainActivity2::class.java)
   val item = items.get(recView.getChildAdapterPosition(it))

   val options = ActivityOptionsCompat.makeSceneTransitionAnimation
   options?.putString("NOM", item.titulo)
   options?.putInt("LOGO", item.foto)

   //Añadimos la información al Intent
   intent.putExtras(options!!)
   startActivity(intent, options)
}
```

- En la segunda actividad recogeremos los datos pasados desde la primera actividad (el nombre del usuario y la imagen), y los mostraremos en pantalla.

```
//Localizamos los controles
Toolbar toolMensaje = (Toolbar) findViewById(R.id.toolc);
ImageView imagLogo = (ImageView)findViewById(R.id.photo);

//Accedemos al Intent que ha originado la primera actividad y recup
Bundle bundle = getIntent().getExtras();

//Mostramos los datos
toolMensaje.setTitle(bundle.getString("NOM"));
imagLogo.setImageResource(bundle.getInt("LOGO"));
```

En Kotlin:

```
val toolMensaje = findViewById<ToolBar>(R.id.toolc)
val imagLogo = findViewById<ImageView>(R.id.photo)
```

```
val bundle = intent.extras

toolMensaje.title= bundle!!.getString("NOM")

imagLogo.setImageResource(bundle!!.getInt("LOGO"))
```

Transición animada entre actividades con elementos compartidos

Consiste en realizar una transición animada entre dos actividades a través de, una vista compartida o componente común en ambas. La animación a realizar estará predefinida por Android.

Para realizar la transición entre dos actividades es necesario:

1. Permitir transiciones en el tema de la aplicación.
2. Especificar las transiciones de los elementos compartidos en el estilo.
3. Definir las transiciones en ficheros .xml, dentro de res/transition.
4. Asignar un nombre común a los elementos compartidos en ambos layouts con el atributo android:transitionName.
5. Usar el método ActivityOptionsCompat.makeSceneTransitionAnimation() para realizar la transición entre actividades.

Veamos los pasos a seguir con un ejemplo:

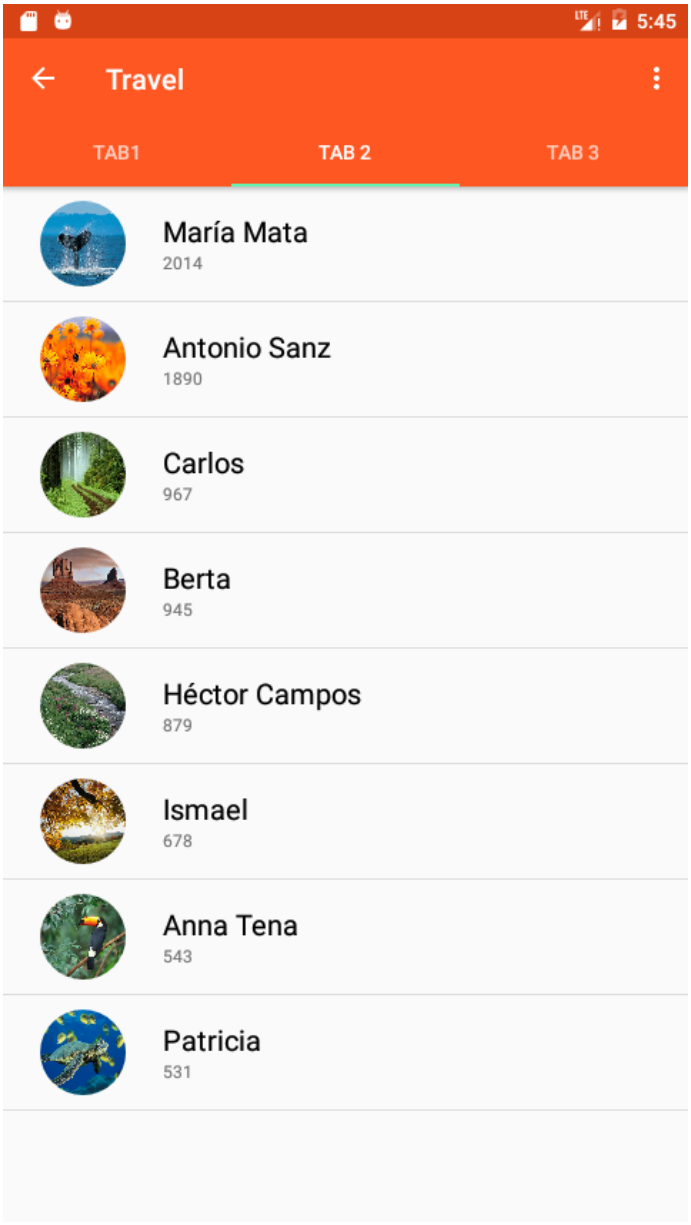
Ejemplo. Transición animada entre actividades con un elemento compartido.

En la primera actividad de nuestro ejemplo, tendremos un ListView donde, en cada item aparecerá un texto (nombre del usuario) y una imagen, y en la segunda actividad aparecerá

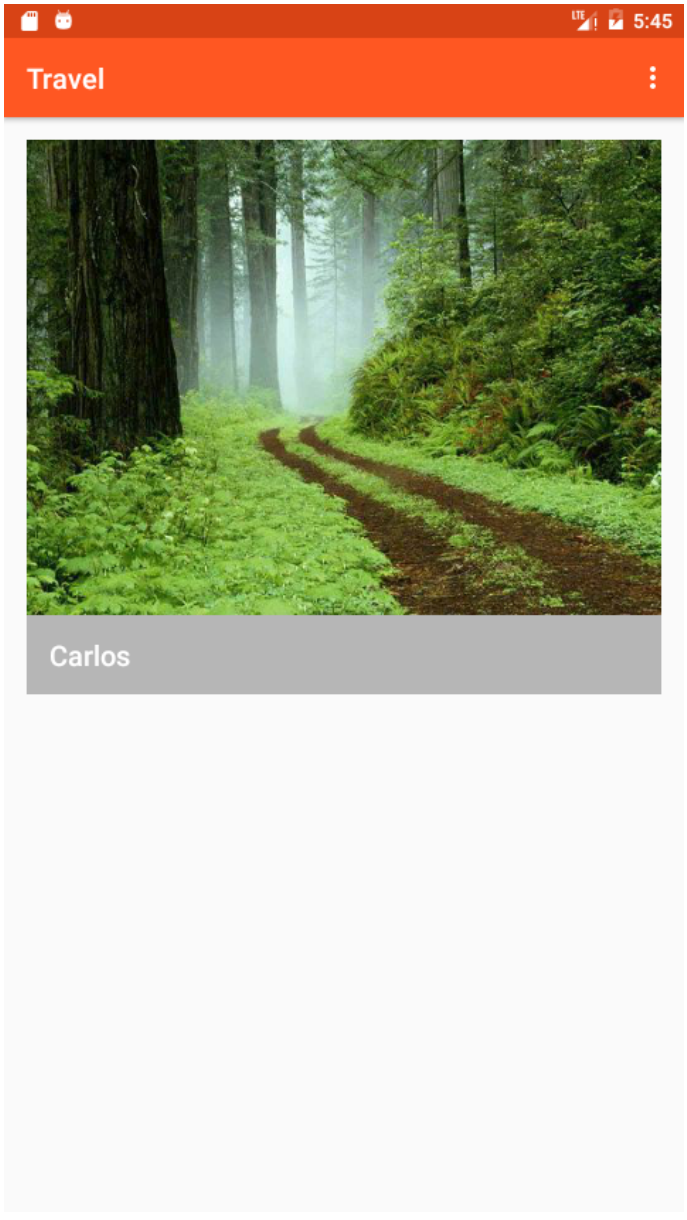
desplegada la misma imagen que teníamos en la primera y su texto correspondiente. Al pulsar sobre un ítem del ListView de la primera escena o actividad, se aplicará una animación únicamente al nombre del usuario (vista compartida) que nos llevará hasta la segunda escena (actividad).

El resultado a obtener será:

Primera actividad (MainActivity.java)



Segunda actividad (MainActivity2.java)



- Diseñamos ambos layouts con un componente compartido, el texto o nombre del usuario. Lo más importante es el nombre que le vamos a dar a la transición entre actividades, ya que cuando compartimos un elemento usando este tipo de transición, las vistas o componentes deben tener el mismo identificador en el parámetro

transitionName.

Layout del ítem del ListView

```
...

<TextView android:id="@+id/idTitulo"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:transitionName="@string/tran_text"
    android:textSize="20sp"
    android:layout_marginLeft="16dp"
    android:layout_gravity="center_vertical"
    android:layout_weight="1"
    android:textColor="#ff000000" />

...
```

```
<androidx.appcompat.w
    android:id="@+id/
    android:layout_he
    android:layout_wi
    android:backgrour
    android:minHeight
    android:transitic
    android:layout_gr
    android:theme="@s
</androidx.appcompat.
```

- Permitimos las transiciones en el tema, y especificamos la transición animada de entrada y salida que se aplicará sobre el elemento compartido en el fichero de estilos (*styles.xml*):

```
<resources>
 <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">

   <!-- enable window content transitions -->
   <item name="android:windowContentTransitions">true</item>

   <!-- specify shared element transitions -->
   <item name="android:windowSharedElementEnterTransition">@transit
   <item name="android:windowSharedElementExitTransition">@transiti
 </style>
</resources>
```

- Definimos las transiciones en *res/transition.* El fichero *change_bounds.xml* se definiría como sigue:

```
<?xml version="1.0" encoding="utf-8"?>

<transitionSet xmlns:android="http://schemas.android.com/apk/res/an
  <changeBounds/>
</transitionSet>
```

- En la primera actividad definimos la transición animada sobre el nombre del usuario, al hacer click sobre un ítem del ListView (lstOpciones).

```
lstOpciones.setOnItemClickListener(new AdapterView.OnItemClickListe
    public void onItemClick(AdapterView<?> a, View v, int position,

        Intent intent = new Intent(this, MainActivity2.class);

        //Animación aplicada sobre el elemento compartido en ambas
        final View sharedtextview = v.findViewById(R.id.idTitulo);
        Pair<View, String> p1 = new Pair(sharedtextview, sharedtext
        Bundle options = ActivityOptionsCompat.makeSceneTransitionA


        AndroidUsers item =((AndroidUsers)a.getItemAtPosition(posit
        options.putString("NOM",item.getTitulo());
        options.putInt("LOGO", item.getLogo());

        //Añadimos la información al Intent
        intent.putExtras(options);
        startActivity(intent, options);
 }
});
```

En Kotlin, sobre el ítem del RecyclerView:

```
adaptador.onClick = {

    val intent = Intent(this, MainActivity2::class.java)

    val sharedtextview: TextView = it.findViewById(R.id.idTitulo)
    val item = items.get(recView.getChildAdapterPosition(it))
```

```
    val p1 = Pair.create<View, String>(sharedtextview, sharedtextvi

    val options = ActivityOptionsCompat.makeSceneTransitionAnimatio
    options?.putString("NOM", item.titulo)
    options?.putInt("LOGO", item.foto)

    //Añadimos la información al Intent
    intent.putExtras(options!!)
    startActivity(intent, options)
}
```

- En la segunda actividad recogeremos los datos pasados desde la primera actividad (el nombre del usuario y la imagen), y los mostraremos en pantalla.

```
//Localizamos los controles
Toolbar toolMensaje = (Toolbar) findViewById(R.id.toolc);
ImageView imagLogo = (ImageView)findViewById(R.id.photo);

//Accedemos al Intent que ha originado la primera actividad y recup
Bundle bundle = getIntent().getExtras();

//Mostramos los datos
toolMensaje.setTitle(bundle.getString("NOM"));
imagLogo.setImageResource(bundle.getInt("LOGO"));
```

1.4 Transiciones entre fragments

Transición animada entre fragments

La transición animada entre fragments es muy similar a la que acabamos de ver sobre actividades.

Los tres primeros pasos son los mismos que en las actividades, solo cambiaría el último paso:

- Permitimos las transiciones en el tema de la aplicación.

```
<resources>
 <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
   <!-- enable window content transitions -->
   <item name="android:windowContentTransitions">true</item>
 </style>
</resources>
```

- Asignamos un nombre común a los elementos compartidos en ambos layouts con el atributo android:transitionName.

layout/fragment1.xml

```
...
<ImageView
android:id="@+id/idLogo"
android:layout_margin="16dp"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:scaleType="centerCrop"
android:transitionName="tranphoto" />
...
```

layout/fragment2.xml

```
 <ImageView
android:id="@+id/idLogo"
android:layout_width="match_
android:layout_height="300dp
android:transitionName="tran
android:scaleType="centerCro
```

- Definimos las transiciones en *res/transition.* El fichero fade.*xml* se definiría como sigue:

```
<?xml version="1.0" encoding="utf-8"?>

<transitionSet xmlns:android="http://schemas.android.com/apk/res/an
  <fade/>
</transitionSet>
```

- Desde el Fragment1.java cargamos el Fragment2.java e incluimos el elemento compartido como parte de FragmentTransaction:

```
final Fragment2 fragment2 = new Fragment2();

Transition fade = TransitionInflater.from(getActivity()).inflateTra


// Define enter and return transitions only for shared element

fragment2.setSharedElementEnterTransition(fade);
fragment2.setSharedElementReturnTransition(fade);

ImageView idlogo = findViewById(R.id.idLogo);

getActivity().getSupportFragmentManager().beginTransaction()
 .replace(R.id.content_frame, fragment2)
 .addToBackStack(null)
 .addSharedElement(idlogo, idlogo.getTransitionName())
 .commit();
```

1.5 Transiciones entre escenas

Transición entre escenas

Consiste en realizar una transición animada y personalizada entre dos escenas (layouts) de una misma actividad.

Para hacer esto definiremos los siguientes elementos:

- Crearemos un layout para cada escena.
- En el layout principal de la actividad *activity_main.xml*, tendremos un componente donde cargaremos las diferentes escenas. Al iniciar la aplicación, inflaremos el layout correspondiente a la primera escena (*scene_main1.xml*).
- Definiremos la transición animada a realizar sobre los componentes en *res/transition*.
- Finalmente, daremos funcionalidad a la aplicación saltando entre escenas con TransitionManager.go()

Veamos los pasos a seguir con un ejemplo:

Ejemplo. Transición animada entre escenas.

Vamos a crear una aplicación con dos escenas o layouts, como los indicados a continuación. Al pulsar sobre la imagen de la primera escena, se aplicará una animación personalizada sobre los componentes de dicho layout que nos llevará hasta la segunda escena.

El resultado a obtener será:

Primera escena (scene_main1.xml) Segunda escena (scene_main2.xml)

Los pasos a seguir son:

- Crearemos los dos layouts indicados anteriormente (*scene_main1.xml* y *scene_main2.xml)*.
- El layout principal de la actividad *activity_main.xml*, contendrá un componente donde cargaremos las diferentes escenas. Al iniciar la aplicación, inflaremos el layout correspondiente a la escena 1(*scene_main1.xml*).

```
<LinearLayout>

    ...
```

```
        <FrameLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:id="@+id/scene_root">
            <include layout="@layout/scene_main1" />
        </FrameLayout>

        ...
</LinearLayout>
```

- A continuación, definimos la transición animada a realizar en *res/transition/info.xml.* Los componentes que presentarán animación en el cambio de escenas serán:
    - el ImageView --> animación changeBounds
    - el TextView del título, del texto descriptivo y el ChipGroup --> animación slide
    - el FAB --> animación fade

```
<?xml version="1.0" encoding="utf-8"?>
<transitionSet xmlns:android="http://schemas.android.com/apk/res/an
    <changeBounds android:duration="1000">
        <targets>
            <target android:targetId="@+id/imagen"/>
        </targets>
    </changeBounds>
    <slide android:duration="1000" android:slideEdge="bottom">
        <targets>
            <target android:targetId="@id/titulo"/>
            <target android:targetId="@id/description"/>
            <target android:targetId="@+id/choice_chip_group"/>
       </targets>
    </slide>
    <fade android:duration="1000" android:fadingMode="fade_out">
       <targets>
            <target android:targetId="@id/fab"/>
       </targets>
    </fade>
```

```
</transitionSet>
```

- Finalmente le daremos funcionalidad a la aplicación. Al hacer click sobre el FAB de la primera escena saltaremos a la segunda escena donde se contraerá la imagen y aparecerá un texto descriptivo del lugar representado en la imagen.

```
final ImageView imagen = findViewById(R.id.imagen);

imagen.setOnClickListener(new View.OnClickListener() {

   @Override
   public void onClick(View v) {

     Transition transicion = TransitionInflater.from(MainActivity.t
     TransitionManager.go(Scene.getSceneForLayout((ViewGroup) findV

    }

});
```