Laboratorio 6.2

Apellidos: Moreno Vera
Nombres: Felipe Adrian
Código: 20120354I
Asignatura: Programación en Dispositivos Móviles (CC481)

2016 - I

Indice

Actividad 1

1. Lo primero es crear nuestro diseño de pantalla tal y como pone la imagen. El fichero XML de main tendrá el RelativeLayout (por tanto utilice las propiedades de este Layout) con:

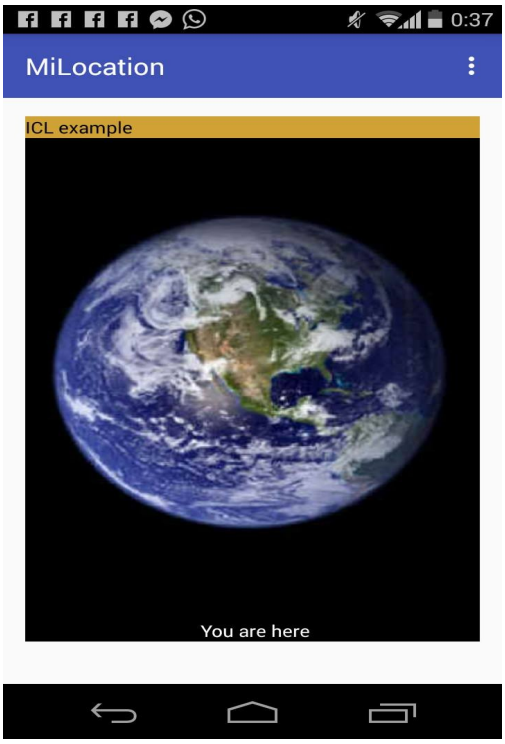
1. Un ImageView con la propiedad (vea la carpeta descargada).
android:background=”@drawable/background”
2. Un TextView con la propiedad:
android:text=”@string/info_text”

3. Para ver la interactividad de la aplicación hay que cambiar el idioma en preferencias para ver su efecto.

Solución:

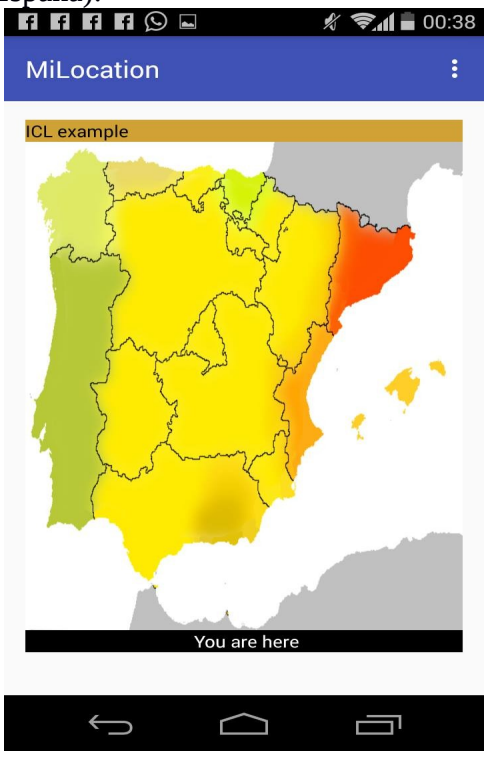
Mi celular por defecto esta en el idioma Japonés, por lo que aparece el mundo, y para hacer pruebas, cambie a español (España) y a francés (Francia), especifico francia, pues hay 3 opciones de francés en mi android.

Celular en idioma Japonés:



Celular en idioma Francés:

Celular en idioma Español(España):



Actividad 2

1. Creamos un nuevo proyecto.
2. Creamos una clase Java que extienda del control que queramos.
public class ExampleCustom extends EditText
3. En los tres constructores creados automáticamente llamamos simplemente a la clase padre.

```
public ExtendedEditText(Context context, AttributeSet attrs, int defStyle){
super(context, attrs,defStyle);
inicializacion();
}
public ExtendedEditText(Context context, AttributeSet attrs) {
super(context, attrs);
inicializacion();
}
public ExtendedEditText(Context context) {
super(context);
inicializacion();
}
```

4. Lo primero a realizar es definir el objeto Paint para dibujar (Verifique de la teoría). ¿Que realiza la línea marcada en negrita?

```
private void inicializacion(){
Paint p1 = new Paint(Paint.ANTI_ALIAS_FLAG);
p1.setColor(Color.BLACK);
p1.setStyle(Style.FILL);
}
Paint p2 = new Paint(Paint.ANTI_ALIAS_FLAG);
p2.setColor(Color.WHITE);
p2.setTextSize(20);
escala = getResources().getDisplayMetrics().density;
```

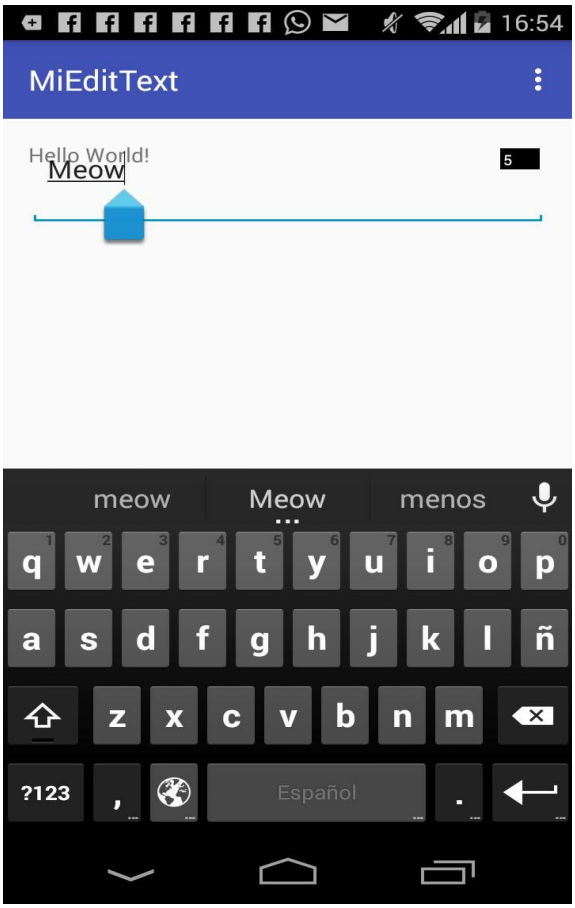
Establece la metrica a mostrar, demarca los limites del canvas.

5. Acto seguido añadimos nuestro método onDraw(). Explique los parámetros de drawText y drawRect.

```
@Override
public void onDraw(Canvas canvas) {
super.onDraw(canvas);
canvas.drawRect(this.getWidth()-30*escala, 5*escala,
this.getWidth()-5*escala, 20*escala, p1) ;
}
canvas.drawText("" + this.getText().toString().length(),
this.getWidth()-28*escala, 17*escala, p2);
```

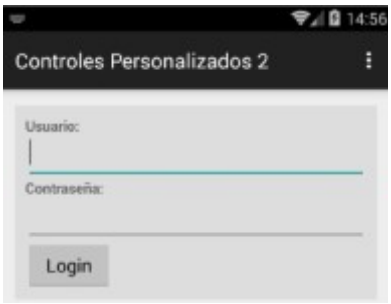
6. Por último añadimos al layout nuestro Custom View y ejecutamos la aplicación.

Solución:



Actividad 3

1. Creamos un nuevo proyecto.
2. Creamos nuestro segundo archivo XML con el nombre login.xml con el diseño mostrado en la siguiente imagen.
NOTA: Tenga en cuenta que el Button de login está dentro de un LinearLayout (horizontal) que a su vez se encuentra dentro del LinearLayout (vertical) original. Como hemos visto anteriormente para que aparezca el mensaje "login correcto/incorrecto" a la iquierda del Button irá un TexView




```
<LinearLayout>
        <EditText .../>
        <TextView .../>
        <EditText .../>
        <TextView .../>
        <LinearLayout ....>
                <Button .../>
                <TextView .../>
        </LinearLayout>
</LinearLayout>
```

3. Como hemos dicho vamos a personalizar nuestro evento. Para ello en este caso vamos a crear una inteface Java llamada OnLoginListener. Describa la funcionalidad de dicha interface una vez implementado el punto 4.

```
public interface OnLoginListener {
void onLogin(String usuario, String password);
}
```

4. Creamos ahora nuestra clase Java login.java. Esta clase heredará de LinearLayout por la vista anteriormente creada.

1. Creamos nuestros atributos.

```
private OnLoginListener listener;
private EditText txtUsuario;
private EditText txtPassword;
private Button btnLogin;
private TextView lblMensaje; //el mensaje de CustomView
```

2. Creamos los constructores

```
public ControlLogin(Context context) {
super(context);
inicializar();
}
public ControlLogin(Context context, AttributeSet attrs) {
super(context, attrs);
inicializar();
TypedArray a = getContext().obtainStyledAttributes(attrs,
R.styleable.ControlLogin);
String textoBoton = a.getString(R.styleable.ControlLogin_login_text);
btnLogin.setText(textoBoton);
a.recycle();
}
```

3. Explique lo que realiza el siguiente método.

```
private void inicializar() {
String infService = Context.LAYOUT_INFLATER_SERVICE;
LayoutInflater li = (LayoutInflater)
getContext().getSystemService(infService);
li.inflate(R.layout.login, this, true);
txtUsuario = (EditText)findViewById(R.id.TxtUsuario);
txtPassword = (EditText)findViewById(R.id.TxtPassword);
btnLogin = (Button)findViewById(R.id.BtnAceptar);
lblMensaje = (TextView)findViewById(R.id.LblMensaje);
a.recycle();
}
```

Define todos los contenidos de nuestro Linear Layout al momento de iniciar el login.

4. Por último personalizamos el evento. Explique la funcionalidad de cada método.

```
public void setOnLoginListener(OnLoginListener l) {
listener = l;
}
private void asignarEventos(){
```

```
btnLogin.setOnClickListener(new OnClickListener() {
public void onClick(View v) {
listener.onLogin(txtUsuario.getText().toString(),
txtPassword.getText().toString());
}
});
}
public void setMensaje(String msg){
lblMensaje.setText(msg);
}
```

5. En nuestro XML principal tendremos que añadir con nuestra clase anterior añadiendo el login.java, como hemos visto en el ejercicio anterior.

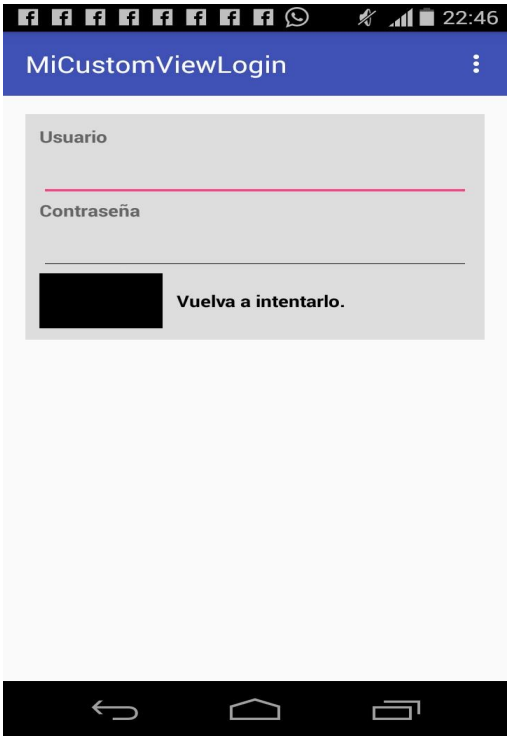
```
<paqueteJava.Login
android:id="@+id/CtlLogin"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:background="#DDDDDD" />
```

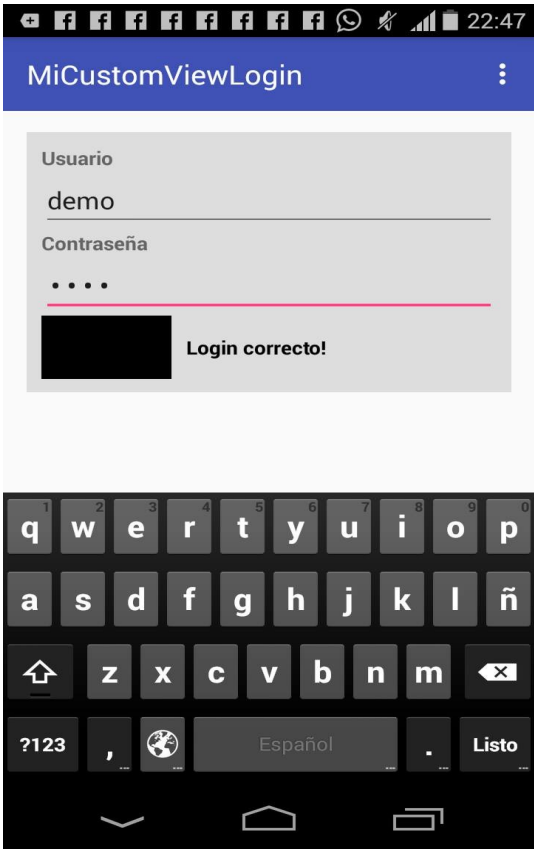
6. Por último, añadimos toda la funcionalidad a nuestra actividad principal en el método onCreate.

```
setContentView(R.layout.activity_main);
ctlLogin = (Login)findViewById(R.id.CtlLogin);
ctlLogin.setOnLoginListener(new OnLoginListener(){
@Override
public void onLogin(String usuario, String password){
if (usuario.equals("demo") && password.equals("demo"))
ctlLogin.setMensaje("Login correcto!");
else
ctlLogin.setMensaje("Vuelva a intentarlo.");
}
});
```

Mi botón aparece en negro ;(lo cambie a texto blanco, sin embargo, no pasa nada ;(

Solución:


22:46
MiCustomViewLogin
Usuario
Contraseña
Vuelva a intentarlo.


22:47
MiCustomViewLogin
Usuario
demo
Contraseña
Login correcto!
Español
Listo

Actividad 4

1. Vamos a realizar el diseño de nuestra aplicación (de la Activity main) tal y como muestra la imagen, con un ImageView y dos Button. Un Button será de rotación y el otro de escalado de la imagen.

1. Declaramos nuestra variable ImageView.
private ImageView animationTarget;

2. Dentro del método onCreate capturamos la imagen de nuestra variable.
AnimationTarget = (ImageView) this.findViewById(R.id.imagen);

3. A continuación creamos el método onClick para rotar la imagen. En el código vemos que se carga un objeto Animator que carga la animación. Explique dicho método.

```
Public void rota (View v){
Animator animation = AnimatorInflater.loadAnimator
(this, R.anim.rotate_around_center_point);
animation.setTarget(animationTarget);
animation.start();
}
```

Este método llama a la clase Animator dándole el ambito y a nuestro xml, luego comienza con la animación

4. El método de “escala” será igual, pero llamando a otro XML para el escalado. Le pondremos de nombre “escala.xml”

3. Veamos ahora el código XML. Como se ha visto en el código se llama a la carpeta /res/anim. En ella vamos a tener dos ficheros XML

1. "rotate_around_center_point.xml" con el siguiente código. La propiedad que realiza la animación es "android:propertyName". Explique las otras propiedades

```
<objectAnimator xmlns:android="http://schemas.android.com/...."
        android:duration="1000"        tiempo de duracion( milisegundos )
        android:valueFrom="0"          posicion rotada inicial
        android:valueTo="360"          posicion rotada final
        android:valueType="floatType"         tipo de valor
        android:propertyName="rotation"      tipo de propiedad a realizar por Animator
        android:repeatCount="0"/>              numero de repeticiones
```

2. "escala.xml". En este caso tendremos otro ObjectAnimator. Explique las propiedades.

```
<objectAnimator xmlns:android="http://schemas.android.com..."
        android:duration="1000"          tiempo de duracion( milisegundos )
        android:valueFrom="1"            tipo de escala inicial
        android:valueTo="0.5"             tipo de escala final
        android:valueType="floatType"        valor a deolver
        android:propertyName="scaleX"       tipo de propiedad a realizar por Animator
        android:repeatCount="1"              numero de repeticiones
        android:repeatMode="reverse"/>
```

4. En la práctica hemos visto como iniciar un objeto Animator desde XML. Como hemos estado viendo este tipo de recursos se puede realizar mediante Java o XML. A continuación exponemos el Java. Pruébelo y comente su ejecución.

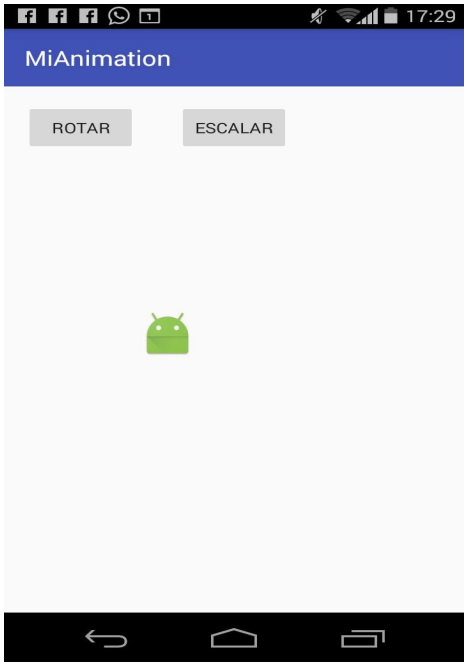
1. El código Java para escala sería:

```
ObjectAnimator scaleXAnimator = ObjectAnimator.ofFloat
(animationTarget, "scaleX", 0.5f);
scaleXAnimator.setRepeatMode(ValueAnimator.REVERSE);
scaleXAnimator.setRepeatCount(1);
scaleXAnimator.setDuration(1000);
scaleXAnimator.start();
```

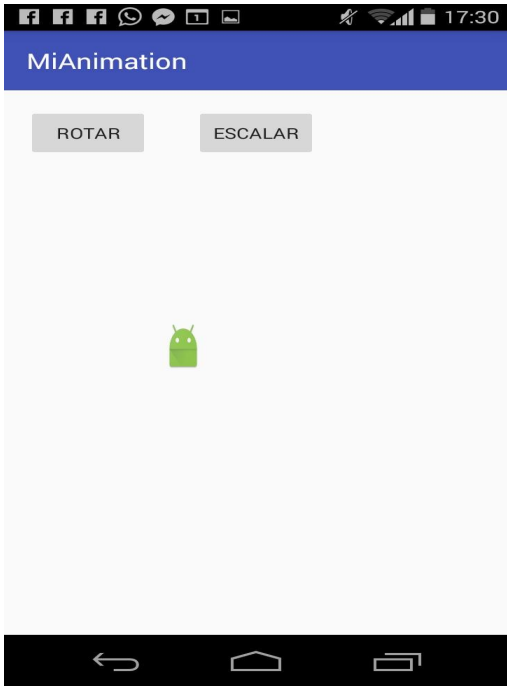
2. El código Java para rotar sería:

```
ObjectAnimator scaleXAnimator = ObjectAnimator.ofFloat
(animationTarget, "rotation", 360f);
scaleXAnimator.setRepeatCount(0);
scaleXAnimator.setDuration(1000);
scaleXAnimator.start();
```

Solución: Este es el normal



Este es después de presionar el escala, disminuye su escala horizontalmente



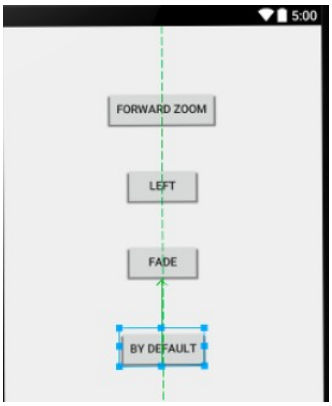
La parte de rotar, si funciona, pero es demasiado rapido para tomarle foto.

Actividad 5

1. Creamos un nuevo proyecto que tendrá dos Activities.
2. El diseño Layout de la primera Activity será como la imagen. Recuerde darle propiedad onClick a los Button.

1. El diseño *Layout* de la primera *Activity* será como la imagen. Recuerde darle propiedad *onClick* a los *Button*.

2. El diseño *Layout* de la segunda *Activity*.



3. A continuación vamos a crear los diferentes *XML* de cada transición. Explique sus propiedades:

1. Para una transición FADE, el XML de entrada (*fade_in.xml*) es:

```
<!--?xml version="1.0" encoding="UTF-8"?-->
<alpha xmlns:android="http:·.."
        android:duration="800"   duracion de cambio
        android:fromalpha="0.0"   tipo de alpha
       android:interpolator="@android:anim/accelerate_interpolator"
        android:repeatcount="0" android:toalpha="1.0"> repetidor
</alpha>
```

Y el de salida (*fade_out.xml*)

```
<!--?xml version="1.0" encoding="UTF-8"?-->
<alpha xmlns:android="http:·…"
        android:duration="800"
        android:fromalpha="1.0"
       android:interpolator="@android:anim/accelerate_interpolator"
        android:repeatcount="0"
        android:toalpha="0.0">
</alpha>
```

2. Para una transición LEFT, el XML de entrada (*left_in.xml*) es:

```
<!--?xml version="1.0" encoding="UTF-8"?-->
<translate xmlns:android="http:·.."
        android:duration=”500"
        android:fromxdelta="100%p"  tipo delta de modificacion
       android:toxdelta="0"
        android:interpolator="@android:anim/linear_interpolator">
</translate>
```

Y el de salida (*left_out.xml*)

```
<!--?xml version="1.0" encoding="UTF-8"?-->
< translate xmlns:android="http:·…"
        android:duration="500"
        android:fromxdelta="0"
       android:toxdelta="-100%p"
        android:interpolator="@android:anim/linear_interpolator">
</ translate>
```

3. Para una transición RIGTH, el XML de entrada (*right_in.xml*) es:

```
<!--?xml version="1.0" encoding="UTF-8"?-->
<translate xmlns:android="http:·.."
        android:duration=”500"
        android:fromxdelta="-100%p"
       android:toxdelta="0"
        android:interpolator="@android:anim/linear_interpolator">
</translate>
```

Y el de salida (*right_out.xml*)

```
<!--?xml version="1.0" encoding="UTF-8"?-->
< translate xmlns:android="http:·…"
        android:duration="500"
        android:fromxdelta="0"
       android:toxdelta="100%p"
        android:interpolator="@android:anim/linear_interpolator">
```

</ translate>

4. Para una transición ZOOM BACK, el XML de entrada (*zoom_back_in.xml*) es:

```
<!--?xml version="1.0" encoding="UTF-8"?-->
<scale xmlns:android="http://..."
        android:duration="600"
        android:fromxscale="0.7"
        android:fromyscale="0.7"
        android:interpolator="@android:anim/decelerate_interpolator"
        android:pivotx="50%p"
        android:pivoty="50%p"
        android:toxscale="1" android:toyscale="1">
</scale>
```

Y el de salida (*zoom_back_out.xml*)

```
<!--?xml version="1.0" encoding="UTF-8"?-->
<set xmlns:android="http:·.."
        android:interpolator="@android:anim/decelerate_interpolator"
android:zadjustment="top">
<scale
        android:fromxscale="1"
        android:fromyscale="1"
        android:pivotx="50%p"
        android:pivoty="50%p"
        android:toxscale="0.7"
        android:toyscale="0.7"
        android:duration="600">

<alpha
        android:fromalpha="1.0"
        android:toalpha="0"
        android:duration="600">
        </alpha></scale></set>
```

5. Para una transición ZOOM FORDWARD, el XML de entrada (*zoom_forward_in.xml*) es:

```
<!--?xml version="1.0" encoding="UTF-8"?-->
<scale xmlns:android="http:·..."
        android:duration="600"
        android:fromxscale="1.5"
        android:fromyscale="1.5"
        android:interpolator="@android:anim/decelerate_interpolator"
        android:pivotx="50%p"
        android:pivoty="50%p"
        android:toxscale="1.0" android:toyscale="1.0"></scale>
```

Y el de salida (*zoom_forward_out.xml*)

```
<!--?xml version="1.0" encoding="UTF-8"?-->
<set xmlns:android="http:·.."
        android:interpolator="@android:anim/decelerate_interpolator"
        android:zadjustment="top">

<scale
        android:duration="600"
        android:fromxscale="1.0"
        android:fromyscale="1.0"
        android:pivotx="50%p"
        android:pivoty="50%p"
        android:toxscale=".7"
        android:toyscale=".7">

<alpha
        android:duration="600"
        android:fromalpha="1.0"
        android:toalpha="0">
        </alpha></scale></set>
```

5. Pasamos por último a ver los Java. De *Main Activity* simplemente añadimos todos eventos *onClick*.

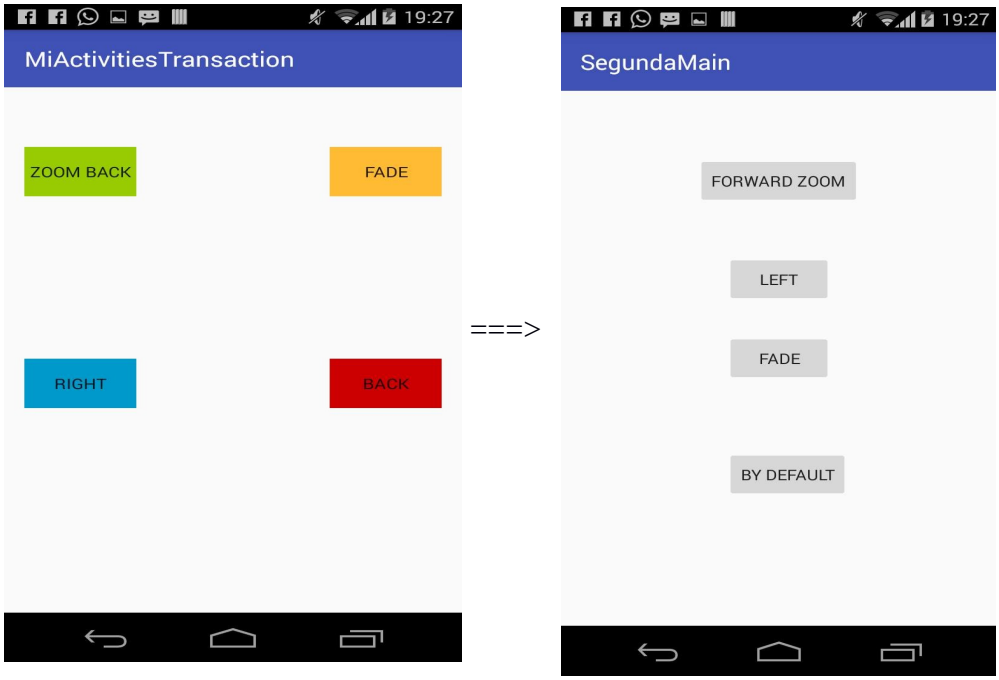
```
public void forwardZoom(View button){
    startActivity(new Intent(this, SecondActivity.class));
    overridePendingTransition(R.anim.zoom_forward_in,
      R.anim.zoom_forward_out);
 }
public void left(View button){
    startActivity(new Intent(this, SecondActivity.class));
    overridePendingTransition(R.anim.left_in, R.anim.left_out);
}
public void fade(View button){
    startActivity(new Intent(this, SecondActivity.class));
    overridePendingTransition(R.anim.fade_in, R.anim.fade_out);
}
public void byDefault(View button){
    startActivity(new Intent(this, SecondActivity.class));
}
```

De la segunda *Activity* igual que el anterior los evento correspondientes.

```
public void zoomBack(View button){
    startActivity(new Intent(this, MainActivity.class));
    overridePendingTransition(R.anim.zoom_back_in,
```

```
        R.anim.zoom_back_out);
}
public void fade(View button){
        startActivity(new Intent(this, MainActivity.class));
        overridePendingTransition(R.anim.fade_in, R.anim.fade_out);
                }
public void right(View button){
        startActivity(new Intent(this, MainActivity.class));
        overridePendingTransition(R.anim.right_in, R.anim.right_out);
}
public void back(View button){
        super.onBackPressed();
}
```

Solución:



Link del github con los códigos del laboratorio:

https://github.com/Jenazad/PDM/tree/master/Laboratorio_6

Referencias

http://stackoverflow.com/questions/3166501/getting-the-screen-density-programmatically-in-android
http://developer.android.com/intl/es/guide/topics/graphics/2d-graphics.html
http://www.sgoliver.net/blog/interfaz-de-usuario-en-android-controles-personalizados-i/
http://www.sgoliver.net/blog/interfaz-de-usuario-en-android-controles-personalizados-ii/
http://www.programcreek.com/java-api-examples/android.content.res.TypedArray
http://developer.android.com/intl/es/guide/topics/resources/animation-resource.html