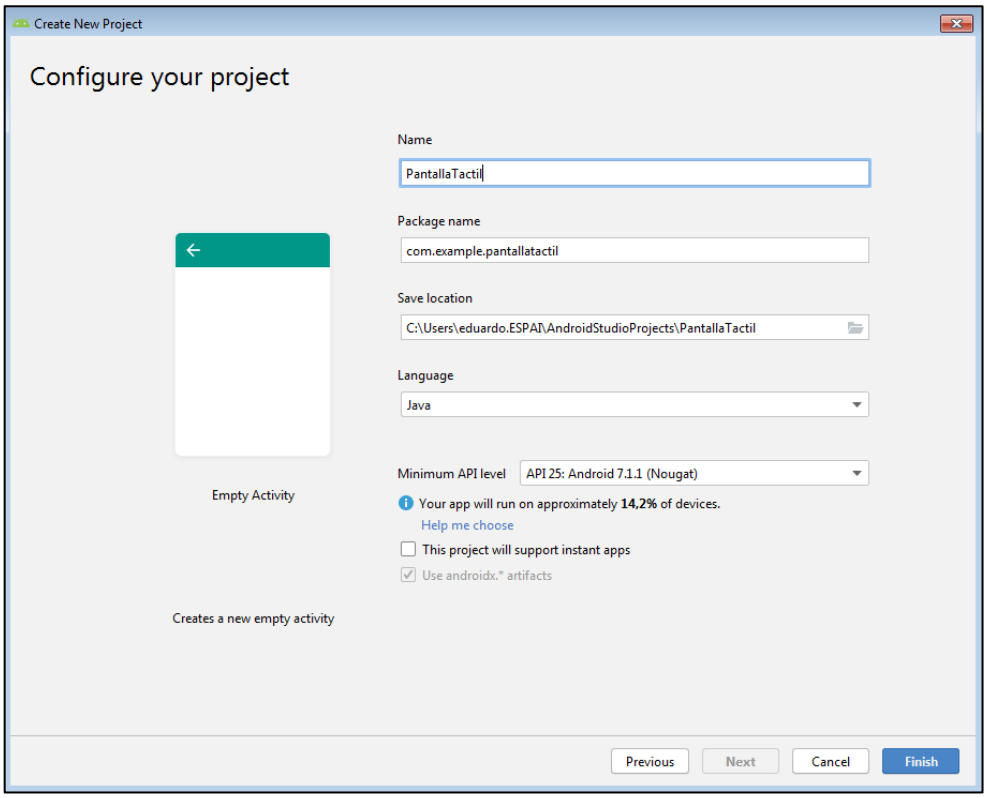
PRACTICA 6: PANTALLA TACTIL

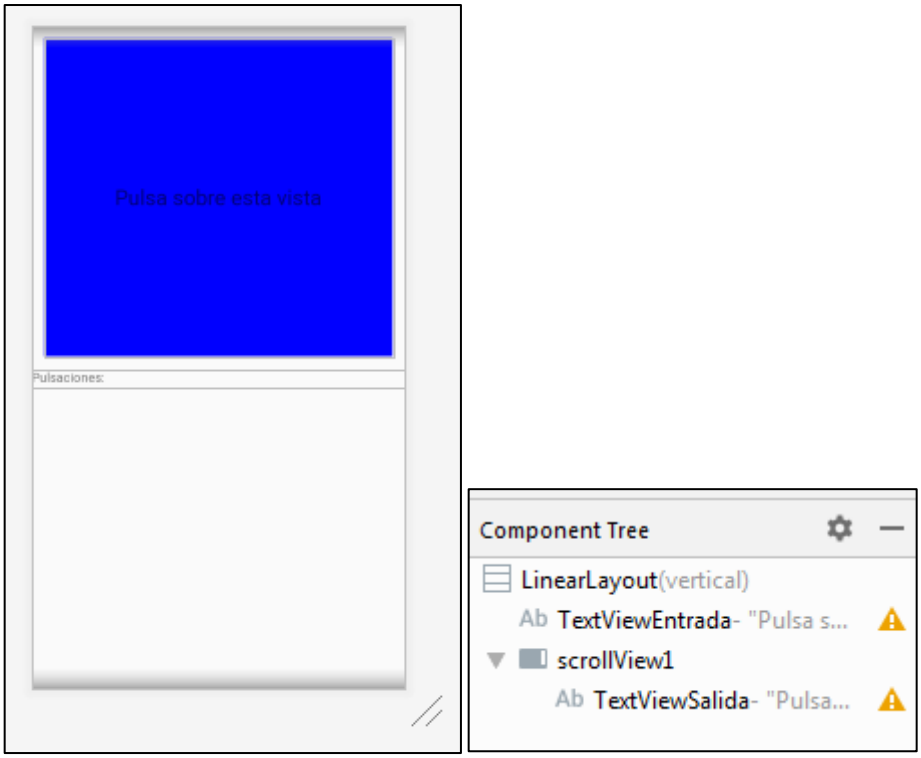
Parte I: Uso de la pantalla táctil

En este ejercicio se mostrará cómo podemos capturar los eventos procedentes de la pantalla táctil. También se aprovechará para repasar otros conceptos, como: Creación de Layouts y herramientas de revisión de código en Eclipse.

Paso 1. Crea un nuevo proyecto con nombre *PantallaTactil*.



Paso 2. Modifica el Layout principal para que tenga una apariencia similar a la siguiente. De esta forma practicarás la creación de Layouts. A la derecha se muestra la estructura de vistas que contiene.



Paso 3. Una posible solución se muestra a continuación:

activity_main.xml

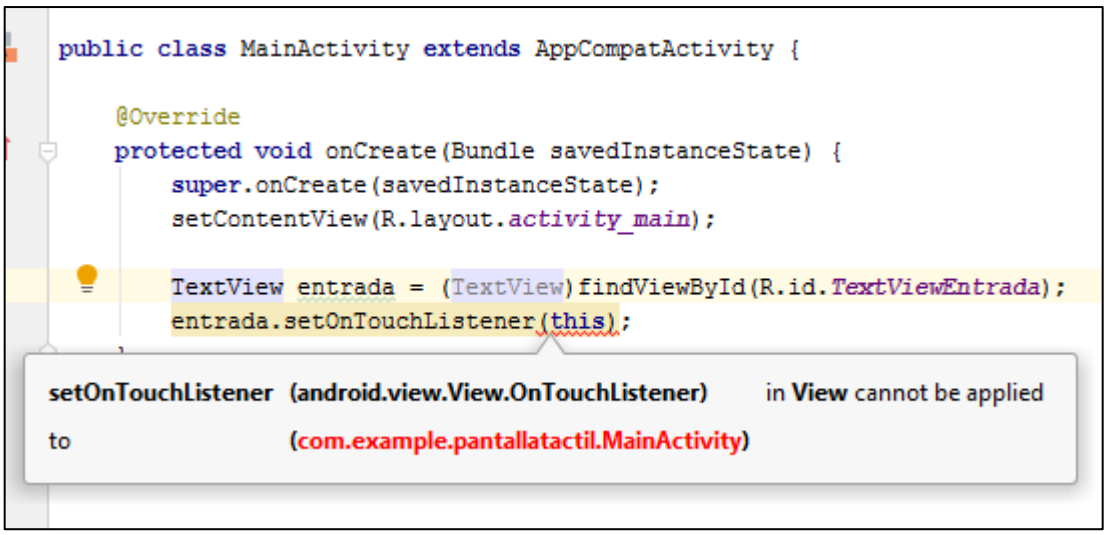
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView
        android:id="@+id/TextViewEntrada"
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:text="Pulsa sobre esta vista"
        android:gravity="center"
        android:background="#0000FF"
        android:layout_margin="2mm"
        android:textSize="10pt"/>
    <ScrollView
        android:id="@+id/scrollView1"
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="1" >
        <TextView
            android:id="@+id/TextViewSalida"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:text="Pulsaciones:"/>
    </ScrollView>
</LinearLayout>
```

Paso 4. Introduce las siguientes dos líneas al final del método onCreate():

```
public class MainActivity extends AppCompatActivity {

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

        TextView entrada = (TextView)findViewById(R.id.TextViewEntrada);
        entrada.setOnTouchListener(this);
    }
}
```

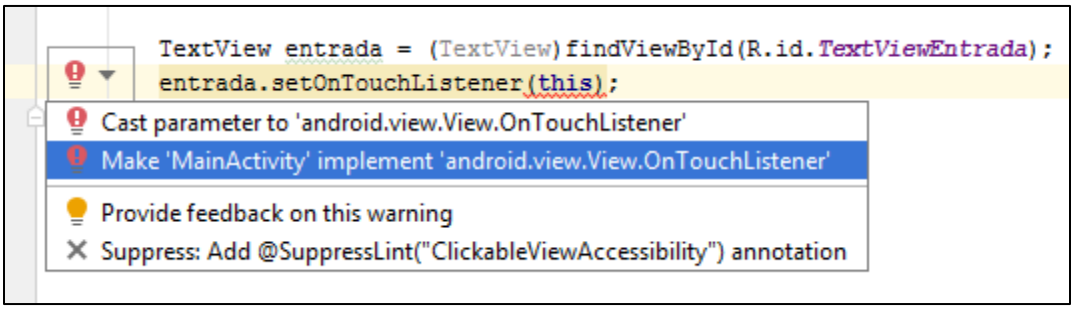
Paso 5. Pulsa *Alt* + *Intro* para añadir los *imports*.

Paso 6. Observa como el método setOnTouchListener está marcado como erróneo. Si pones el cursor encima, te indicará que el parámetro de este método (this) es de la clase MainActivity, y es necesario que sea de tipo OnTouchListener.

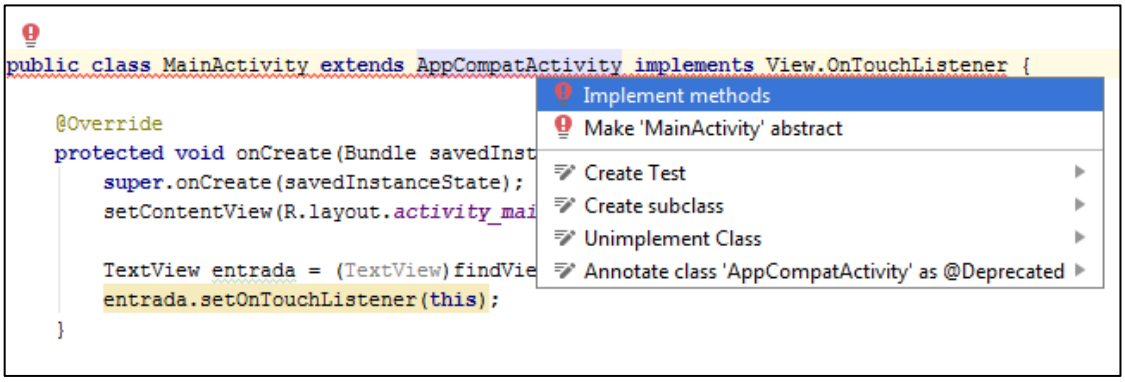


Paso 7. Para evitar el error te mostrará una lista de posibles soluciones. Selecciona la última "Make MainActivity implement 'android.view.View.OnTouchListener'" de esta forma implementaremos este interfaz y nuestra clase podrá ser considerada de este tipo. La declaración de la clase cambiará a:

public class MainActivity **extends** Activity **implements** OnTouchListener {



Paso 8. Se ha solucionado el problema anterior, pero ha aparecido otro. Ahora, MainActivity está marcada como errónea. El problema consiste en que estamos diciendo que implementamos el interfaz OnTouchListener pero no hemos implementado ninguno de los métodos de este interfaz.



Paso 9. Para evitar el error selecciona en la lista de posibles soluciones: "Add unimplemented methods" de esta forma se añadirán todos los métodos necesarios de este interfaz. La declaración de la clase cambiará a:

```
public boolean onTouch(View arg0, MotionEvent arg1) {
  //TODOAuto-generated method stub
  Return false;
}
```

```
public class MainActivity extends AppCompatActivity implements View.OnTouchListener {

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

        TextView entrada = (TextView)findViewById(R.id.TextViewEntrada);
        entrada.setOnTouchListener(this);
    }

    @Override
    public boolean onTouch(View view, MotionEvent motionEvent) {
        return false;
    }
}
```

Paso 10. Reemplaza el nombre de los parámetros por otros más expresivos. Por ejemplo: arg0 por vista y arg1 por evento.

Paso 11. Observa como este método ha de devolver un parámetro. Actualmente es false, que significa que no nos hemos hecho cargo de la pulsación, el sistema seguirá pasando este evento a otras vistas. En este caso el LinearLayout que contiene la vista. Cámbialo a true, para que el sistema no siga propagando este evento.

```
@Override
public boolean onTouch(View view, MotionEvent motionEvent) {
    return true;
}
```

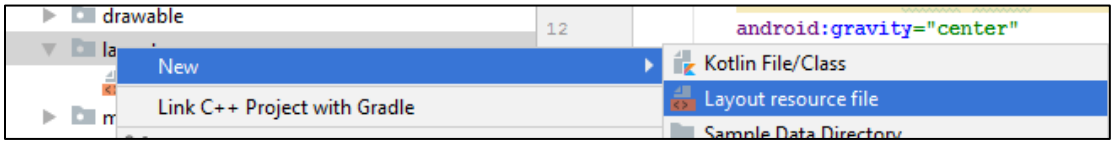
Paso 12. Agrega las siguientes líneas:

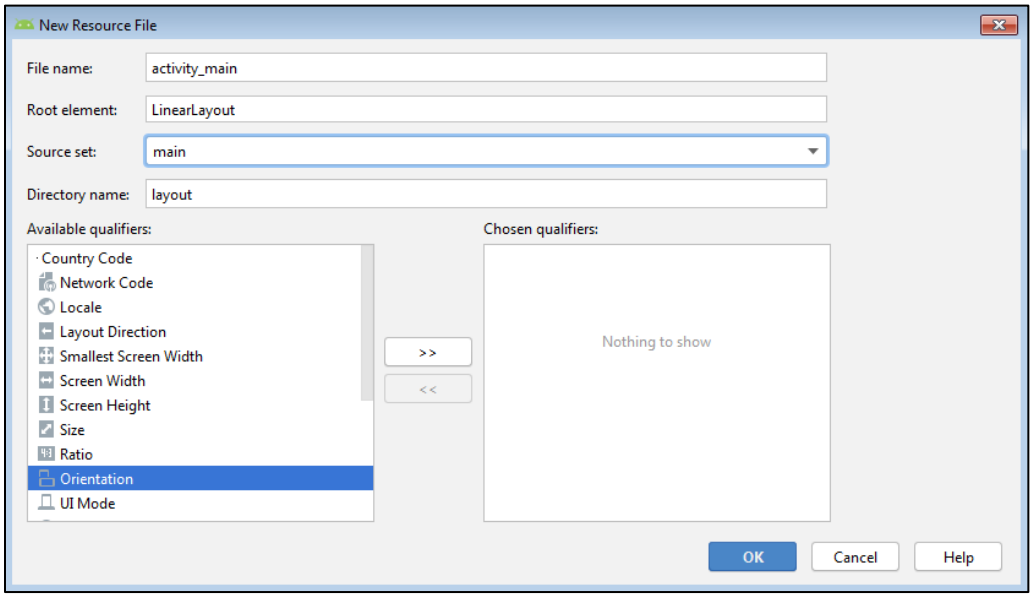
```
@Override
public boolean onTouch(View view, MotionEvent motionEvent) {
    TextView salida = (TextView) findViewById(R.id.TextViewSalida);
    salida.append(motionEvent.toString()+"\n" );

    return true;
}
```

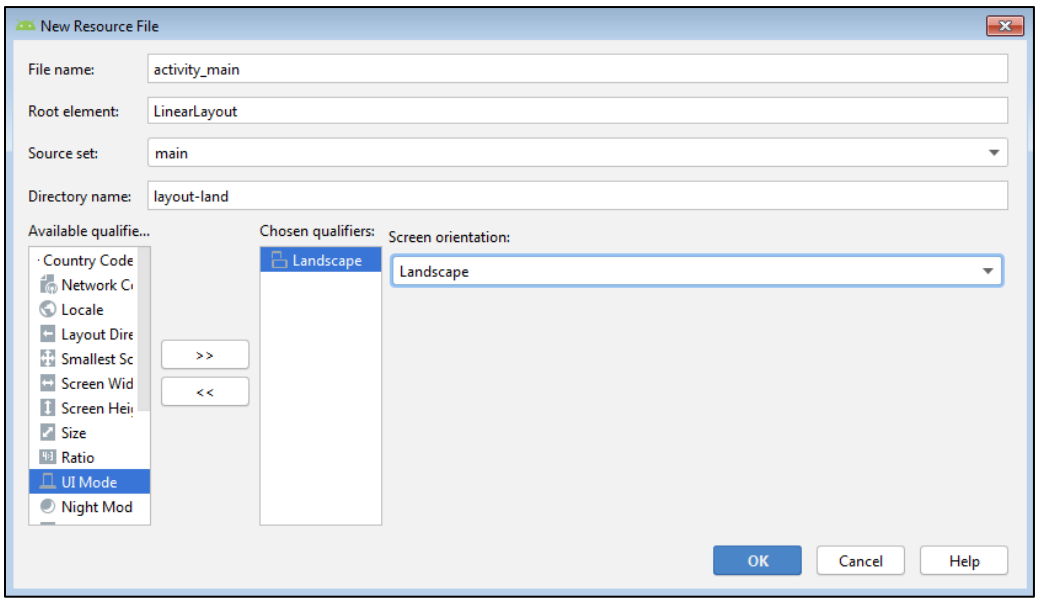
Paso 13. Ejecuta el proyecto y verifica el resultado.

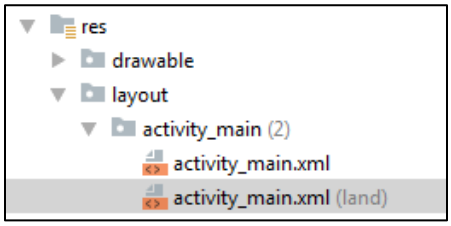
action=0 significa que se ha pulsado sobre la pantalla, action=1 significa que se ha soltado y action=2 que se está desplazando el dedo. (Estos tres valores corresponden con las constantes MotionEvent.ACTION_DOWN, MotionEvent.ACTION_UP y MotionEvent.ACTION_MOVE)

Paso 14. Modifica el proyecto para que cuando el móvil se ponga en apaisado el Layout que se visualice sea:





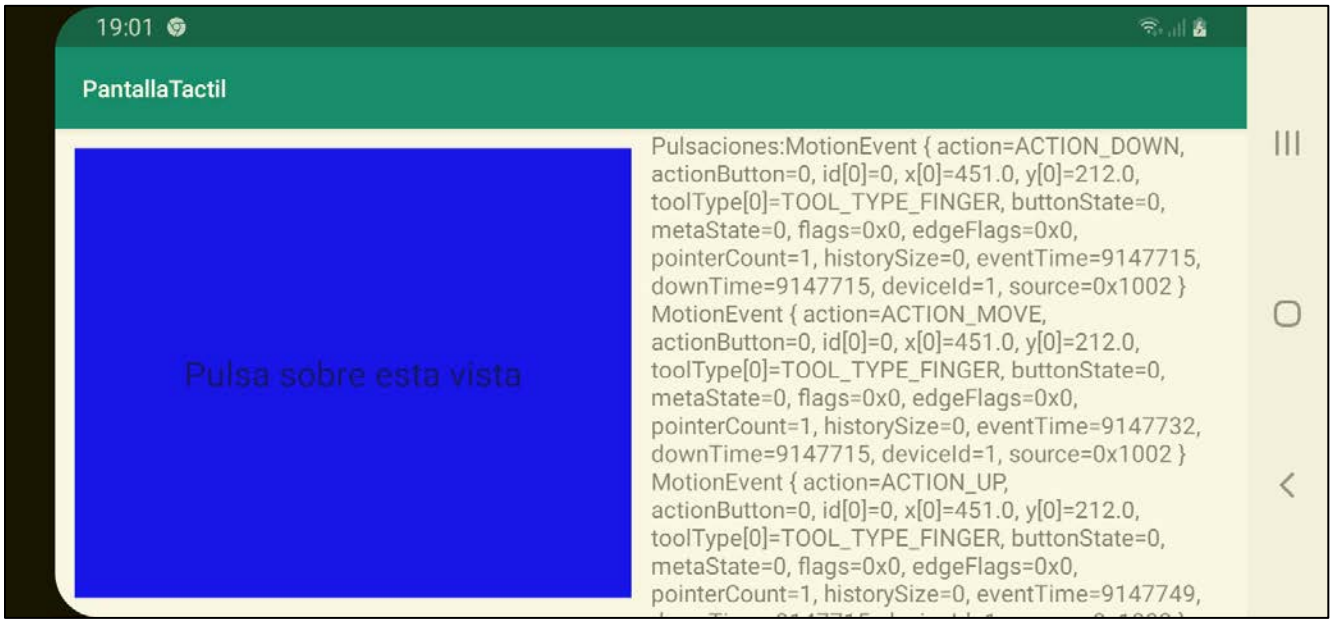




land\activity_main.xml

```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="horizontal" >
    <TextView
        android:id="@+id/TextViewEntrada"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:text="Pulsa sobre esta vista"
        android:gravity="center"
        android:background="#0000FF"
        android:layout_margin="2mm"
        android:textSize="10pt"/>
    <ScrollView
        android:id="@+id/scrollView1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1" >
        <TextView
            android:id="@+id/TextViewSalida"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:text="Pulsaciones:"/>
    </ScrollView>
</LinearLayout>
```

Paso 16. Verifica el resultado en un dispositivo real.



Paso 17. No todas las pantallas táctiles soportan los métodos getPression() y getSize(). Prueba con tu terminal si lo soporta y en tal caso observa el rango de valores que obtienes.

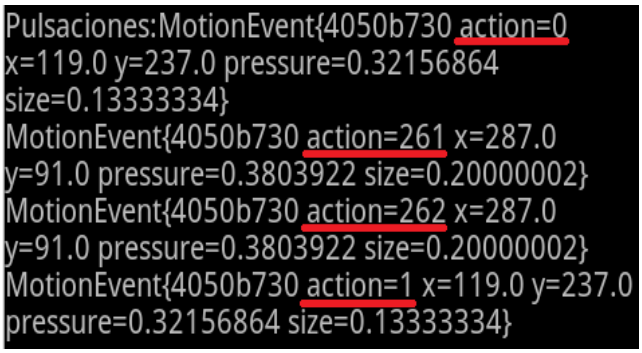
Paso 18. Conéctate a www.androidcurso.com y localiza el ejercicio que acabas de realizar. Comprueba los valores obtenidos con otros terminales y comparte los valores obtenidos por tu terminal.

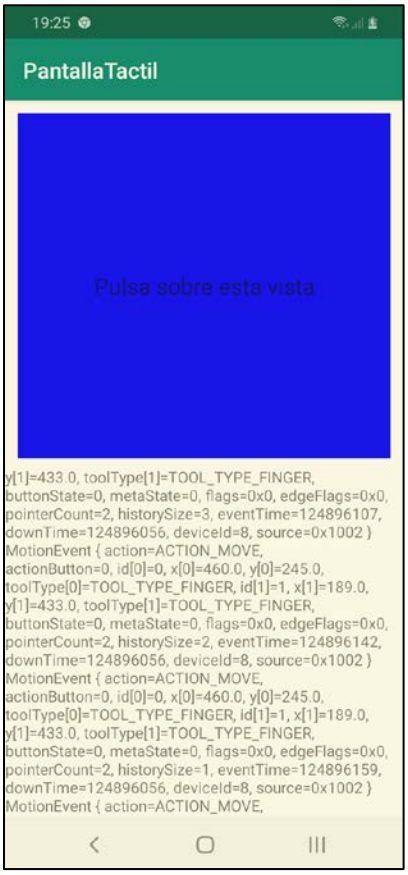
Parte II: Uso de la pantalla táctil multi-touch

Paso 1. Ejecuta el ejercicio anterior en un dispositivo real con capacidad de multi-touch (si no dispones de uno te será imposible realizar este ejercicio).

```
public class MainActivity extends AppCompatActivity implements View.OnTouchListener {

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

        TextView entrada = (TextView) findViewById(R.id.TextViewEntrada);
        entrada.setOnTouchListener(this);

        if (getPackageManager().hasSystemFeature(PackageManager.FEATURE_TOUCHSCREEN_MULTITOUCH)){
            Toast.makeText( context: MainActivity.this,  text: "El dispositivo tiene capacidad tactil con multitouch",
                    Toast.LENGTH_LONG).show();
        }else{
            Toast.makeText( context: MainActivity.this,  text: "El dispositivo no tiene capacidad tactil con multitouch",
                    Toast.LENGTH_LONG).show();
        }

    }
```

Paso 2. Pulsa simultáneamente con dos dedos en la pantalla: Si lo haces sin desplazar los dedos recibirás 4 eventos. Los dos primeros por las pulsaciones de cada dedo y los dos siguientes cuando se levanten. El resultado puede ser simular al siguiente:





Paso 3. Como puedes ver cuando hay más de un puntero en pantalla la acción resulta compleja de interpretar. Veremos cómo hacerlo a continuación.

Paso 5. Reemplaza la siguiente línea del método onTouch()

salida.append(evento.toString()+"\n");por:

```
@Override
public boolean onTouch(View view, MotionEvent motionEvent) {

    TextView salida = (TextView) findViewById(R.id.TextViewSalida);
    //salida.append(motionEvent.toString()+"\n" );

    String acciones[] = { "ACTION_DOWN", "ACTION_UP", "ACTION_MOVE", "ACTION_CANCEL",
            "ACTION_OUTSIDE", "ACTION_POINTER_DOWN", "ACTION_POINTER_UP" };
    int accion = motionEvent.getAction();
    int codigoAccion = accion & motionEvent.ACTION_MASK;
    salida.append(acciones[codigoAccion]);
    for (int i = 0; i < motionEvent.getPointerCount(); i++) {
        salida.append(" puntero:" + motionEvent.getPointerId(i) +
                " x:" + motionEvent.getX(i) + " y:" + motionEvent.getY(i));
    }
    salida.append("\n");

    return true;
}
```

Paso 6. Para visualizar cada posible acción hemos creado un array con sus nombres. A continuación averiguamos la acción en la variable accion. Esta nueva acción la ha podido hacer cualquier puntero de los activos o uno nuevo. A partir de la versión 2.0 en esta variable se codifica simultáneamente el código de la acción (8 bits menos significativos) e índice de puntero que la ocasiona (siguientes 8 bits). Para obtener esta información por separado puedes utilizar el siguiente código:

```
int codigoAccion = accion & MotionEvent.ACTION_MASK;
int iPuntero = (accion & MotionEvent.ACTION_POINTER_ID_MASK) >> MotionEvent.ACTION_POINTER_ID_SHIFT;
```

Paso 7. A partir de la versión 2.2 (API level 8) las dos últimas constantes quedan obsoletas y se definen otras dos equivalentes cuyos nombres son más adecuados:

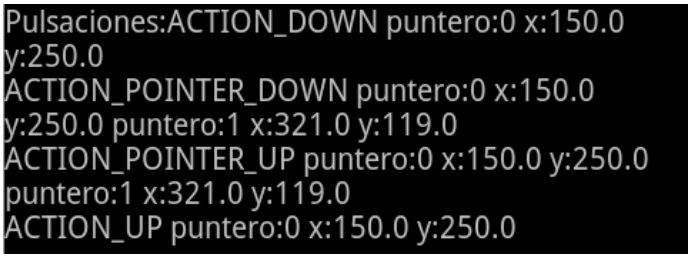
```
int codigoAccion = accion & motionEvent.ACTION_MASK;
int iPuntero = (accion & MotionEvent.ACTION_POINTER_INDEX_MASK) >>
        MotionEvent.ACTION_POINTER_INDEX_SHIFT;
```

Paso 8. Una vez obtenido el código de la acción mostramos su nombre en la vista salida. Luego hacemos un bucle para mostrar información de todos los punteros activos. El método getPointerCount() nos permite averiguar su número. Vamos a recorrer los punteros activos con la variable i. Al principio de este apartado vimos una serie de métodos para averiguar información sobre el puntero (getX(), getSize(),…). A partir de la versión 2.0 estos métodos siguen dándome información sobre el primer puntero activo que se pulsó, pero ahora también disponemos de los mismos métodos pero indicando un índice de puntero (getX(i), getSize(i),…) para averiguar información del resto de punteros.

```
for (int i = 0; i < evento.getPointerCount(); i++) {
    salida.append(" puntero:" + evento.getPointerId(i) + " x:" +
                  evento.getX(i) + " y:" + evento.getY(i));
}
```

Paso 9. El método getPointerId(**int** indice) nos permite averiguar el identificador del puntero. No hay que confundir el índice de puntero con su identificador. El índice se asigna en función del orden en que fueron pulsados. El índice cero siempre es el más antiguo. El índice de un puntero decrece a medida que los punteros anteriores a él dejan de estar activos. Por el contrario, el identificador de un puntero es asignado cuando se crea y permanece constante durante toda su vida. Nos será muy útil para seguir la pista de un determinado puntero. El método findPointerIndex(**int** id) nos permite averiguar el índice de un puntero a partir de su identificador.

Paso 10. Ejecuta de nuevo el proyecto y vuelve a pulsar con dos dedos. El resultado ha de ser similar al siguiente:



Paso 11. Prueba con otras combinaciones de pulsaciones e investiga la relación entre el índice y el id de puntero.

getPointerId(**int** indice) nos da el identificador del puntero. Es asignado cuando se crea y permanece constante durante toda su vida.

findPointerIndex(**int** id) nos da el índice de un puntero a partir de su identificador. Se asigna en función del orden en que fueron pulsados. El índice cero siempre es el más antiguo. El índice de un puntero decrece a medida que los punteros anteriores a él dejan de estar activos

Paso 12. Modifica el programa para que además se muestre en cada evento, el índice de puntero que lo ocasionó.

```
@Override
public boolean onTouch(View view, MotionEvent motionEvent) {

    TextView salida = (TextView) findViewById(R.id.TextViewSalida);
    //salida.append(motionEvent.toString()+"\n" );

    String acciones[] = { "ACTION_DOWN", "ACTION_UP", "ACTION_MOVE", "ACTION_CANCEL",
            "ACTION_OUTSIDE", "ACTION_POINTER_DOWN", "ACTION_POINTER_UP" };
    int accion = motionEvent.getAction();

    int codigoAccion = accion & motionEvent.ACTION_MASK;
    int iPuntero = (accion & MotionEvent.ACTION_POINTER_INDEX_MASK) >>
            MotionEvent.ACTION_POINTER_INDEX_SHIFT;

    salida.append(acciones[codigoAccion]);
    for (int i = 0; i < motionEvent.getPointerCount(); i++) {
        salida.append(" puntero:" + motionEvent.getPointerId(i) +
                " indice: "+motionEvent.findPointerIndex(motionEvent.getPointerId(i)) +
                " x:" + motionEvent.getX(i) + " y:" + motionEvent.getY(i));
    }
    salida.append("\n");

    return true;
}
```