Capítulo 6
Sensores y otro hardware

Práctica 6.1: Vibración

La posibilidad de vibrar ha formado parte del mundo de los dispositivos móviles desde sus inicios. Android nos proporciona un API para controlar la vibración desde su primera versión, a través de una clase de nombre android.os.Vibrator.

El acceso a una instancia de esa clase se realiza a través del contexto ("entorno de la aplicación"). En concreto, se considera un servicio del sistema (igual que lo son, por ejemplo, el servicio de alarma o el de localización). Por tanto, debemos pedirle al contexto actual dicha instancia usando el método getSystemService().

Afortunadamente, las actividades son contextos, por lo que no tendremos que preocuparnos en conseguirlo. En cualquier método de nuestra actividad podremos hacer:

```
Vibrator vibrator;
vibrator = (Vibrator) getSystemService(VIBRATOR_SERVICE);
```

Los métodos de la clase son:

- **void vibrate(long miliseconds):** hace que vibre durante los milisegundos indicados. No es bloqueante.
- **void cancel():** detiene la vibración en marcha.
- **void vibrate(long[] pattern, int repeat):** vibra con un patrón. El array del primer parámetro representa el patrón a reproducir, y va alternando el número de milisegundos en "reposo" y en "activo", empezando en reposo.

Además, podemos pedir que se repita el efecto en ciclo. Para eso, en el segundo parámetro se debe especificar la posición del patrón a la que volver (siendo 0 la primera). Si no se quiere ejecución en un bucle, se debe especificar -1.

Es importante tener en cuenta que la repetición del patrón siempre empieza "en pausa". Por tanto, para que el ciclo funcione como se espera, el segundo parámetro debería ser siempre par (donde están los tiempos de pausa en el array), pues de otro modo convertiremos lo que en la primera reproducción fue tiempo de espera en tiempo de vibración y viceversa.

- **boolean hasVibrator():** está disponible a partir del API 11 (Android 3.0). Indica si el dispositivo puede o no vibrar.

Para probar la vibración, vamos a hacer una pequeña aplicación que reproduce una "canción" y pregunta cual es.

- Crea un nuevo proyecto y llámalo Vibración.
- En el método onCreate(), obtén el objeto Vibrator solicitándoselo a la actividad, y guárdalo en un atributo privado del objeto, para evitar tener que recuperarlo contínuamente. Activa la vibración durante 100 milisegundos a modo de "bienvenida".

```
@Override
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_vibracion);
        _vibrator = (Vibrator) getSystemService(VIBRATOR_SERVICE);
        _vibrator.vibrate(100);
} // onCreate

private Vibrator _vibrator;
```

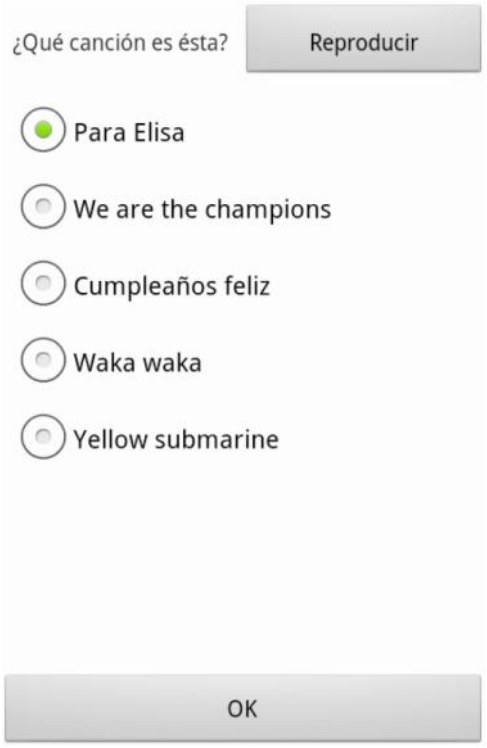
Ten en cuenta que esto solicitará la vibración del móvil también cuando esté activa la actividad y se gire. ¿Cómo lo evitarías?

- Prueba la aplicación[1]. Fallará.
- Activar la vibración se considera una actividad privilegiada, por lo que debemos solicitar permiso de uso. Modifica el fichero de manifiesto y añade:

```
<uses-permission android:name="android.permission.VIBRATE"/>
```

También lo puedes hacer desde la parte visual de Android Studio, abriendo el fichero, y en la pestaña "Permissions", añadiendo uno de tipo Uses Permission, con la etiqueta anterior (android.permission.VIBRATE).

- Vuelve a probar la aplicación. Ahora sí debería funcionar.
- Manipula el layout para conseguir la ventana de la figura.



[1] Como podrás suponer, el emulador de AVD no emula la vibración, por lo que tendrás que usar un dispositivo físico.

```
<RelativeLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:padding="10dp">

        <TextView
                android:id="@+id/tvCancion"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:padding="8dp"
                android:layout_alignBaseline="@+id/bReproducir"
                android:layout_alignParentLeft="true"
                android:text="@string/queCancionEs"/>

        <Button
                android:id="@+id/bReproducir"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_toRightOf="@+id/tvCancion"
                android:layout_alignParentRight="true"
                android:padding="8dp"
                android:text="@string/reproducir"
                android:onClick="onReproducir"
                tools:context=".Vibracion" />

        <ScrollView
                android:id="@+id/scrollView"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_below="@id/bReproducir"
                android:layout_above="@+id/bAceptar">

                <RadioGroup
                        android:id="@+id/rgOpciones"
                        android:layout_width="fill_parent"
                        android:layout_height="wrap_content"
                        android:padding="8dp">

                <RadioButton
                        android:id="@+id/ParaElisa"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:text="@string/ParaElisa"
                        android:checked="true"/>

                <RadioButton
                        android:id="@+id/champions"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:text="@string/champions"/>

                <RadioButton
                        android:id="@+id/cumple"
```

```
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/cumple"/>

        <RadioButton
                android:id="@+id/wakaWaka"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/wakaWaka"/>

        <RadioButton
                android:id="@+id/yellowSubmarine"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/yellowSubmarine"/>
        </RadioGroup>
    </ScrollView>

    <Button
        android:id="@+id/bAceptar"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:onClick="onAceptar"
        android:text="@android:string/ok"/>
</RelativeLayout>
```

- Añade las cadenas en el fichero de recursos que necesites.
- Define el método onReproducir() llamado al pulsar el botón de reproducción. Tendrás que definir el patrón de una "canción" utilizando la duración de las notas de su melodía, e insertando pequeños instantes de pausa entre nota y nota a modo de separadores. Por ejemplo:

```
public void onReproducir(View v) {


    _vibrator.cancel();

    final int corcheaBase = 200;
    final int hueco = 10;
    final int negra = corcheaBase * 2 - hueco;
    final int blanca = corcheaBase * 4 - hueco;
    final int corchea = corcheaBase - hueco;
    long[] patron = {0, // Siempre empezamos con "pausa"

    corchea, hueco, corchea, hueco,
    blanca, hueco, negra, hueco,negra, hueco,
    blanca, hueco + negra + hueco,

    corchea, hueco, corchea, hueco,
    blanca, hueco, negra, hueco,
    negra, hueco, blanca, hueco + negra + hueco,

    corchea, hueco, corchea, hueco,
```

```
        blanca, hueco, negra, hueco,
        negra, hueco, negra, hueco, blanca, hueco,

        corchea, hueco, corchea, hueco,
        blanca, hueco, negra, hueco,
        negra, hueco, blanca
};
_vibrator.vibrate(patron, -1);
} // onReproducir
```

- Ejecuta la aplicación y comprueba que al pulsar el botón se reproduce el patrón.
- Implementa el método onPause() del ciclo de vida de la actividad. Si dejamos de ser la actividad en primer plano deberíamos detener nuestra ejecución, de modo que cancelaremos la vibración que pudiéramos tener en marcha:

```
public void onPause() {
super.onPause();
_vibrator.cancel();
} // onPause
```

- Implementa el método llamado cuando se pulsa el botón aceptar. Extrae la opción seleccionada por el usuario, comprueba si es o no la esperada, y muestra un mensaje (en un toast) en función de ello. Asegúrate de detener la vibración que pudiéramos tener en marcha.

```
public void onAceptar(View v) {
        _vibrator.cancel();
        RadioGroup rg;
        rg = (RadioGroup) findViewById(R.id.rgOpciones);
        int idRadioButton;
        idRadioButton = rg.getCheckedRadioButtonId();
        int idString;
        if (idRadioButton == R.id.cumple)
                idString = R.string.acertaste;
        else
                idString = R.string.fallaste;
        Toast.makeText(this, idString, Toast.LENGTH_SHORT).show();
} // onAceptar
```

Ejecuta la aplicación y comprueba que funciona como esperas.

Práctica 6.2: Sensores

Los dispositivos móviles van equipados con una creciente cantidad de sensores que les permiten captar información del entorno a través de diferentes magnitudes físicas.

El acceso a los sensores se realiza a través del gestor de sensores, que puede conseguirse a través del contexto, utilizando el mismo método que usamos en la práctica 6.1 para conseguir el acceso a la capacidad de vibración:

```
SensorManager sm;
sm = (SensorManager) getSystemService(SENSOR_SERVICE);
```

Una vez que tenemos el gestor de sensores, podemos hacer un recorrido sobre todos ellos para averiguar qué capacidades tiene nuestro dispositivo.

En esta práctica vamos a mostrar en una etiqueta los nombres de los sensores disponibles.

- Crea un proyecto nuevo y denomínalo Sensores.
- La actividad creada por el asistente nos sirve en este caso para nuestros objetivos, pues nos vamos a limitar a mostrar información y para eso podemos reutilizar la etiqueta. Modifica el layout para asociarle el identificador @+id/sensores de modo que podamos acceder a ella posteriormente.
- En el método onCreate() consigue una referencia al gestor de sensores, y utilizando su método getSensorList() obtén una lista con todos los sensores disponibles. Luego recórrela, pregunta a cada uno su nombre y acumúlalos en una cadena. Termina metiendo dicha cadena en la etiqueta de la ventana:

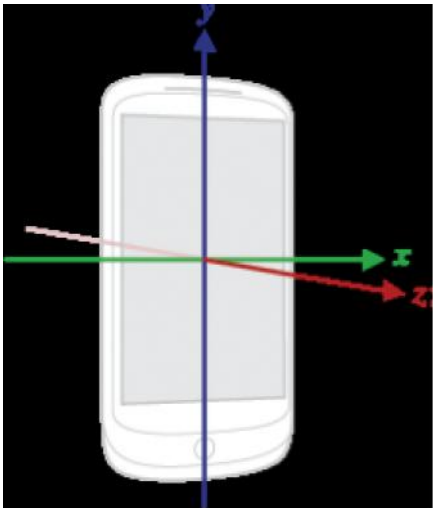
```
@Override
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_sensores);
        SensorManager sm;
        sm = (SensorManager) getSystemService(SENSOR_SERVICE);
        java.util.List<Sensor> listaSensores;
        listaSensores = sm.getSensorList(Sensor.TYPE_ALL);
        StringBuffer sb = new StringBuffer();
        sb.append("Tu móvil tiene ")
                .append(listaSensores.size())
                .append(" sensores\n\n");
        for (Sensor s: listaSensores)
                sb.append(s.getName()).append("\n");
        TextView tv;
        tv = (TextView) findViewById(R.id.sensores);
        tv.setText(sb.toString());
} // onCreate
```

- Ejecuta el programa en un AVD. El resultado es desolador: no simula ningún sensor.
- Ejecútalo en un dispositivo físico y comprueba si los valores devueltos coinciden con lo esperado a partir de su especificación.

En Android se proporciona acceso a una gran cantidad de sensores:

- Acelerómetro, gravedad y aceleración lineal, medidos en m/s^2
- Temperatura ambiente en grados Celsius
- Giróscopo: nos proporciona la velocidad de giro en cada eje, en radianes por segundo
- Nivel de luz medido en "luxes"
- Campo magnético: nos lo proporciona en los tres ejes, y se mide en microteslas
- Presión atmosférica medida en milibares
- Detección de proximidad
- Humedad relativa en porcentaje
- Vector de rotación: indica la orientación del dispositivo

En los sensores en los que es necesario un sistema de referencia, se utiliza el del propio móvil.



Práctica 6.3: Acelerómetro

Seguramente el sensor más conocido sea el del acelerómetro. Detecta la aceleración que sufre el dispositivo en cada momento. Este tipo de sensor llegó al gran público con el mando de la Wii, la consola de Nintendo, aunque llevaba tiempo siendo utilizado en otros lugares, como discos duros o airbags.

A nivel de programación, para todos los sensores tenemos que proceder de la misma manera. En lugar de leer los valores de los sensores, será Android quién nos informe sobre los cambios en sus valores. Por tanto, necesitamos registrarnos como oyentes del sensor en el gestor de sensores. Para eso, utilizaremos el método:

boolean SensorManager::registerListener(SensorListener listener, Sensor sensor, int rate);

donde:

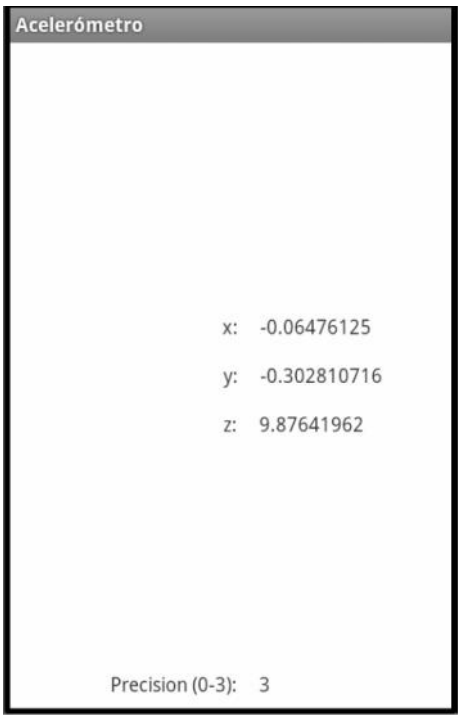
- **listener**: es el objeto en el que queremos recibir las notificaciones declos cambios de los datos del sensor.
- **sensor**: el sensor en el que estamos interesados.
- **rate**: perioricidad con la que queremos ser informados de los cambios. Podemos indicar un valor en microsegundos, aunque lo habitual será especificar una de las constantes definidas en SensorManager **(SENSOR_DELAY_FASTEST, SENSOR_DELAY_GAME, SENSOR_DELAY_NORMAL o SENSOR_DELAY_UI).**

Es importante también pedir que se nos desregistre como oyentes en cuanto dejemos de ser la aplicación en primer plano, dado que los sensores consumen mucha batería, y si hay una aplicación que los utiliza, Android no deja de atenderlos ni siquiera cuando se suspende la pantalla. Por tanto, el procedimiento recomendado es hacer uso del ciclo de vida de la actividad, y realizar el registro en el sensor en el método onResume(), y el desregistro en onPause() usando SensorManager:unregisterListener().

Los oyentes de los sensores deben implementar dos métodos, uno en el que reciben las notificaciones de los cambios leídos, y otro en el que reciben los cambios de precisión. Por comodidad, en este caso no usaremos una clase anónima, sino que directamente la actividad implementará el interfaz del oyente y se registrará a sí misma.

Vamos a terminar de ver los detalles directamente sobre la práctica. Haremos una aplicación que muestre en cuatro etiquetas los valores leídos del acelerómetro y su precisión.

- Crea un proyecto nuevo y denomínalo Acelerómetro.
- Configura el layout para que tenga un aspecto similar al de la figura. En los valores numéricos puedes poner etiquetas sin texto.




```
<RelativeLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent" >

        <TextView
                android:id="@+id/tvX"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_centerHorizontal="true"
                android:layout_above="@+id/tvY"
                android:padding="8dp"
                android:text="@string/x" />
        <TextView
                android:id="@+id/tvValorX"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_toRightOf="@+id/tvX"
                android:layout_alignBaseline="@+id/tvX"
                android:padding="8dp" />
        <TextView
                android:id="@+id/tvY"
                android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:layout_centerHorizontal="true"
            android:layout_centerVertical="true"
            android:padding="8dp"
            android:text="@string/y" />
        <TextView
            android:id="@+id/tvValorY"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_toRightOf="@+id/tvY"
            android:layout_alignBaseline="@+id/tvY"
            android:padding="8dp" />
        <TextView
            android:id="@+id/tvZ"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerHorizontal="true"
            android:layout_below="@id/tvY"
            android:padding="8dp"
            android:text="@string/z" />
        <TextView
            android:id="@+id/tvValorZ"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_toRightOf="@+id/tvZ"
            android:layout_alignBaseline="@+id/tvZ"
            android:padding="8dp"/>
        <TextView
            android:id="@+id/tvPrecision"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignRight="@id/tvZ"
            android:layout_alignParentBottom="true"
            android:padding="8dp"
            android:text="@string/precision" />
        <TextView
            android:id="@+id/tvValorPrecision"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_toRightOf="@+id/tvPrecision"
            android:layout_alignBaseline="@+id/tvPrecision"
            android:padding="8dp"/>
</RelativeLayout>
```

- En la clase de la actividad, define dos nuevos atributos privados. Uno almacenará el gestor de sensores, y el otro guardará el sensor del acelerómetro. Guarda también las etiquetas para poder acceder a ellas directamente sin tener que buscarlas, dado que vamos a tener que actualizarlas a menudo.

```
private SensorManager _sensorManager;
private Sensor _acelerometro;
private TextView _etX;
```

```
private TextView _etY;
private TextView _etZ;
private TextView _etPrecision;
```

En el método onCreate(), consigue una referencia al gestor de sensores y al sensor del acelerómetro. Consigue también las referencias a las etiquetas. No te registres como oyente del acelerómetro aún.

```
@Override
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_acelerometro);

        _sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
        _acelerometro = _sensorManager.getDefaultSensor(
                                Sensor.TYPE_ACCELEROMETER);

        _etX = (TextView) findViewById(R.id.tvValorX);
        _etY = (TextView) findViewById(R.id.tvValorY);
        _etZ = (TextView) findViewById(R.id.tvValorZ);
        _etPrecision = (TextView)
                                findViewById(R.id.tvValorPrecision);
} // onCreate
```

Haz que la clase de la actividad implemente el interfaz necesario para recibir los eventos de los sensores.

```
public class Acelerometro extends Activity implements SensorEventListener {
....
```

Implementa el método onResume() que forma parte del ciclo de vida de la actividad, y en él regístrate como oyente del sensor del acelerómetro:

```
public void onResume() {
        super.onResume();
        _sensorManager.registerListener(this, _acelerometro,
                        SensorManager.SENSOR_DELAY_NORMAL);
}
```

- En el método onPause() desregístrate:

```
public void onPause() {
        super.onPause();
        _sensorManager.unregisterListener(this);
}
```

- Implementa los métodos del interfaz. Actualiza las etiquetas en el que se nos informa de un cambio en el sensor. Del método de cambio de precisión no nos preocuparemos para este sensor.

```
public void onAccuracyChanged(Sensor sensor, int accuracy) {
}
```

```
public void onSensorChanged(SensorEvent event) {
        _etX.setText(""+event.values[0]);
        _etY.setText(""+event.values[1]);
        _etZ.setText(""+event.values[2]);
        _etPrecision.setText(""+event.accuracy);
}
```

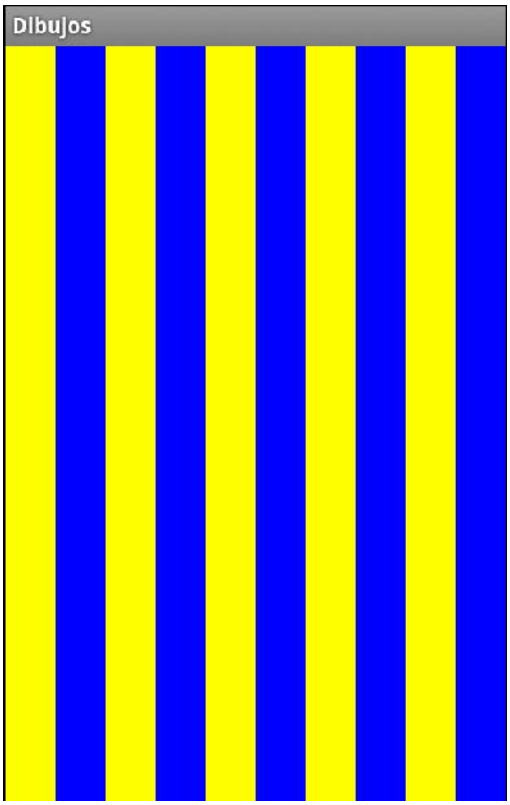
Prueba la práctica en un dispositivo físico. Verás que los valores cambian de acuerdo a su orientación.

Práctica 6.4: Gráficos en Android

En Android tenemos dos maneras de conseguir representar gráficos en nuestras actividades. La primera, y más simple, es hacer uso de los controles (subclases de View) que permiten dibujar imágenes (ImageView e ImageButton). En ese caso, podemos incluirlas directamente en nuestros layouts, por lo que se integran perfectamente con el resto de elementos y resultan cómodas de utilizar. En contra de lo que indica su nombre, pueden encargarse de representar no sólo imágenes sino, de manera general, cualquier subclase de Drawable. Los drawable son recursos, por lo que se integran perfectamente con el modelo de programación y generación de contenido en Android, y no sólo incluyen imágenes, sino también formas geométricas, superposición de figuras, etcétera.

La segunda alternativa es implementar una subclase propia de la clase View, sobreescribir el método onDraw(), que es invocado por Android cuando la vista se tiene que pintar, y programar en ella lo que se considere oportuno. El método recibe un objeto de la clase Canvas, que tiene métodos con operaciones de dibujado.

Vamos a ver un ejemplo muy simple que demuestra esta última posibilidad, para crear una actividad en la que aparecen barras de colores alternos.



- Crea un nuevo proyecto y denomínalo Dibujos.
- Abre el fichero .java. Dentro de la clase de la actividad, inserta una nueva clase (interna) llamada DibujosView que herede de View. Añade el constructor esperado, que deberá recibir un Context. Llama en él a la superclase. En breve añadiremos código adicional.

```
class DibujosView extends View {
	public DibujosView(Context context) {
		super(context);
	}
} // class DibujosView
```

El método onDraw(Canvas) debe ejecutarse lo más rápido posible. El dibujado sobre el canvas requiere el uso de "brochas" (objetos de la clase Paint) que indican una configuración de la operación de dibujado (estilo de relleno, color, estilo del trazo, etcétera). En lugar de crear esos objetos en cada repintado, los vamos a crear en el constructor y los conservaremos como atributos de la clase para reutilizarlos. Define dos nuevos atributos en la clase DibujosView:

```
Paint _brocha1;
Paint _brocha2;
```

Vuelve al constructor y añade el código para crear esas brochas. Una pintará rellenando de amarillo, y la otra de azul:

```
_brocha1 = new Paint();
_brocha1.setColor(0xFFFFFF00);
_brocha1.setStyle(Paint.Style.FILL);

_brocha2 = new Paint();
_brocha2.setColor(0xFF0000FF);
_brocha2.setStyle(Paint.Style.FILL);
```

Implementa el método onDraw() de la clase DibujosView. Será llamado por Android cuando necesite que la vista se repinte. Utilizaremos los objetos Paint anteriores para dibujar las bandas de colores alternos:

```
public void onDraw(Canvas c) {
	float ancho = c.getWidth();
	float alto = c.getHeight();

	float incX = ancho/10;

	Paint brocha = _brocha1;
	for (int i = 0; i < 10; ++i) {
		c.drawRect(i*incX, 0, (i+1)*incX, alto, brocha);
		brocha = (brocha==_brocha1)?_brocha2:_brocha1;
	}
}
```

- La clase DibujosView ya la tenemos preparada. Ahora la clase de la actividad tiene que usar uno de estos objetos como su vista, ignorando el layout en XML del asistente, tal y como ya hicimos en la práctica 2.2:

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(new DibujosView(this));
} // onCreate
```

- Ejecuta el programa.

Práctica 6.5: Multitouch

El multitouch es la detección de varias pulsaciones simultáneas sobre la pantalla. Android informa de los eventos de pulsación a las vistas a través del método onTouchEvent(MotionEvent), y en el parámetro tenemos información sobre el suceso. Esto incluye también el seguimiendo de "todos los dedos", por lo que resulta sencillo, aprovechando los conocimientos de la práctica anterior, hacer una aplicación que nos pinte círculos debajo del lugar donde pulsamos.

- Crea un nuevo proyecto y llámalo Multitouch.
- Igual que antes, tendremos una clase interna que heredará de View. Como atributos guardará una lista estática (con un array y un contador) que mantendrá las posiciones de los dedos (conseguidas en el evento onTouchEvent()) y que serán usados en onDraw() para pintar los círculos. Además, tendremos 10 brochas, una para cada potencial dedo, cada una de un color. En el constructor lo inicializaremos todo:

```
class MultitouchView extends View {
        public MultitouchView(Context context) {
                super(context);
                _brochas = new Paint[10];
                _dedos = new PointF[10];
                int r = 0;
                int g = 0xFF;
                int b = 0;
                for (int i = 0; i < 10; ++i) {
                        _dedos[i] = new PointF();
                        _brochas[i] = new Paint();
                        _brochas[i].setColor(0xFF000000 +
                        (r << 16) + (g << 8) + b);
                        _brochas[i].setStyle(Paint.Style.FILL);
                        r += 20;
                        g -= 20;
                        b += 20;
                } // for
        } // Constructor

        private Paint[] _brochas;
        private PointF[] _dedos;
        private int _numDedos = 0;
} // class MultitouchView
```

En el evento invocado por android ante cada pulsación, actualizaremos la "lista de dedos", e invocaremos a invalidate(), que anula la vista y pide a Android que nos repinte. Añade el código siguiente a la clase MultitouchView.

```
public boolean onTouchEvent(MotionEvent event) {
```

```
        if (event.getAction() == MotionEvent.ACTION_UP)
                _numDedos = 0;
        else {
                _numDedos = event.getPointerCount();
                if (_numDedos > 10)
                        _numDedos = 10;
                for (int i = 0; i < _numDedos; ++i) {
                        _dedos[i].x = event.getX(i);
                        _dedos[i].y = event.getY(i);
                }
        }
        invalidate();
        return true;
}
```

- En el método onDraw(), recorremos "la lista de dedos", y pintamos cada uno con el color de su brocha:

```
public void onDraw(Canvas c) {
        // Ponemos un fondo (borramos lo anterior).
        c.drawColor(0xFFFFFFFF);
        for (int i = 0; i < _numDedos; ++i)
                c.drawCircle(_dedos[i].x, _dedos[i].y, 70, _brochas[i]);
} // onDraw
```

- Ya sólo falta el onCreate() de la actividad, donde añadiremos nuestra vista:

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(new MultitouchView(this));
} // onCreate
```

- Prueba la aplicación en un dispositivo físico, y pulsa con varios dedos en la pantalla. ¡Cada uno saldrá de un color!