Programación de aplicaciones Android: Multimedia

Caso práctico

Ada está contenta del aprendizaje que **María** y **Juan** están realizando sobre el desarrollo de aplicaciones móviles. Ésta es una de las áreas por las que **BK Programación** ha decidido apostar dentro del proceso de modernización que están llevando a cabo. Además del tema de los dispositivos móviles, la empresa también ha decidido acercarse al mundo de la programación de aplicaciones multimedia y del desarrollo de juegos. El caso de los juegos se trata de un tipo de aplicación lo suficientemente específica y compleja como para dedicarle una buena temporada de análisis y estudio. Por eso, **Ada** quiere dedicar un tiempo a echar un vistazo a alguna biblioteca multimedia y a echar un vistazo a las clases y librerías para dibujar con Android.

—Antes de entrar de lleno con el tema de los juegos, quiero que tengáis algunas nociones básicas sobre multimedia, pues es algo que hoy día forma parte de cualquier aplicación y no podemos quedarnos al margen de eso en nuestros programas —les dice **Ada** a **María** y a **Juan** mientras van a buscar a algunos de sus compañeros.

—¿Multimedia? —preguntan ambos al unísono.

—Así es. Aplicaciones que hagan uso de diversos medios de manera simultánea, como imágenes, sonidos, texto, animaciones o cualquier otro. Además, en muchos casos se intenta también proporcionar la máxima interactividad con el usuario —les explica mientras continúan andando.

—¿Y eso cómo lo vamos a poder hacer? ¿Vamos a trabajar con una nueva API? —pregunta ahora **María**.

—No vas por mal camino. Ésa es mi idea. Quiero que os familiaricéis con alguna biblioteca que os facilite la integración multimedia en vuestras aplicaciones -le contesta **Ada**.

Puede decirse que una aplicación multimedia es aquella que es capaz de procesar, reproducir, transmitir, almacenar, etc. contenidos multimedia. Un contenido multimedia está compuesto por diversos medios entre los cuales destacan:

Geral (Licencia Pixabay)

- **Texto:** que puede estar sin formato o con formato (por ejemplo mediante un lenguaje de marcas). El texto además puede ser ser lineal o bien hipertexto (con enlaces a otros textos).
- **Imágenes:** formadas por puntos o píxeles, cada uno de un color. Normalmente obtenidas mediante algún medio de digitalización (fotografía digital, escáner, etc.). Existen múltiples formatos para representar imágenes.
- **Audio:** conteniendo voz, música o cualquier otro tipo de sonido registrado (obtenido mediante algún medio de grabación) o sintetizado (generado por ordenador).
- **Vídeo:** sucesión de imágenes que son proyectadas a gran velocidad dando la sensación de movimiento.
- **Animaciones:** similar al caso del vídeo pero con imágenes generadas por ordenador en lugar de con imágenes captadas de la realidad.
- **Cualquier otro medio** de representación de contenidos que vaya siendo inventado.

En general, para cada tipo de medio no hay una única forma de representar y almacenar la información, sino que existen diversos tipos de formato, cada uno con sus características específicas de estructura, codificación, compresión, etc.

Además del propio formato en el que se representa la información, otro aspecto que debe ser tenido en cuenta es que muchos contenidos multimedia están íntimamente relacionados con el tiempo, en el sentido de que pueden ir cambiando conforme éste avanza (por ejemplo en el caso de audio, vídeo o animaciones). Este tipo de restricciones da lugar a que esos contenidos tengan que ser generados o procesados dentro de unas condiciones muy estrictas de tiempo, para que el flujo de información sea el apropiado, por ejemplo: en el caso de la reproducción de un fragmento de audio o de vídeo.

Otra característica que las aplicaciones multimedia suelen incorporar es la interactividad, es decir, la posibilidad de interactuar con la aplicación para decidir y seleccionar la tarea que deseamos realizar, rompiendo la estructura lineal de la información. De este modo el usuario puede realizar una observación no secuencial de los contenidos en función de las decisiones que vaya tomando durante la ejecución de la aplicación.

Para saber más

Puedes echar un vistazo a los siguientes enlaces para obtener algo más de información sobre el concepto de multimedia:

Multimedia según Wikipedia.

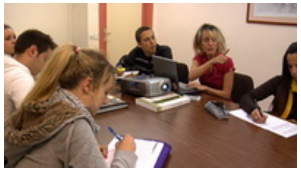
Autoevaluación

Respecto a multimedia, señala la afirmación correcta.

- ○ Las animaciones no se consideran objetos multimedia.
- ○ Un vídeo que capturemos con nuestro dispositivo no es un elemento multimedia.
- ○ El texto puede formar parte de una aplicación multimedia.

1.- Gráficos

Caso práctico



Ada ha buscado a **María** y **Juan** para preguntarles:

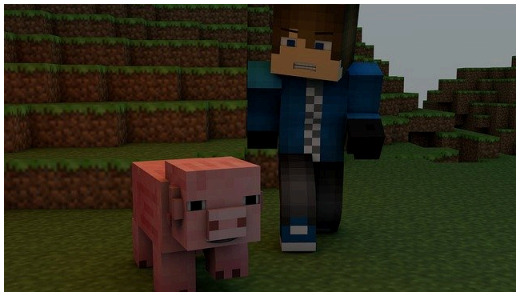
—¿Qué tal andan **Ana** y **Carlos** de conocimientos sobre el tema multimedia? Quizá haría falta hacer un pequeño seminario, ¿no?

—Pues la verdad es que no les vendría mal, y ya de paso a mí tampoco. Tengo que refrescar algunos conceptos —contesta **Juan** interesado.

—De acuerdo, decidles que vengan a la sala de reuniones, vamos a hacer una pequeña presentación sobre el tema antes de empezar a trabajar con una biblioteca concreta —les dice entonces **Ada.**

—Muy bien, vamos allá...

Android proporciona una gran colección de funciones que pueden cubrir prácticamente cualquier necesidad gráfica de una aplicación a través de su API gráfica. Es posible realizar gráficos vectoriales, manipular imágenes, animaciones, operar con texto o gráficos en 3D, etc.



ffstudios (Licencia Pixabay)

Comentaremos las características más significativas del API gráfico de Android, sobre todo las clases más utilizadas para el desarrollo de gráficos en 2D.

Sería imposible abarcar todo lo concerniente a gráficos y multimedia en una unidad, por lo que te recomendamos que consultes el enlace de Android Developers Imágenes y gráficos en Android.

A la hora de crear una aplicación es importante considerar exactamente cuáles serán sus exigencias gráficas, para decantarse por la técnica más adecuada. Por ejemplo, los gráficos y animaciones para una aplicación más bien estática debería implementarse de manera bastante diferente a los gráficos y animaciones para un juego interactivo.

Debes conocer

A continuación, puedes empezar a introducirte en el mundo de los gráficos en Android a través de este interesante vídeo sobre las API gráficas. En él se presentan las clases que nos van a permitir crear y manipular gráficos en Android.

API para gráficos en Android

Resumen textual alternativo

1.1.- Decidiendo dónde dibujar

En las API del **framework Android** encontramos un conjunto específico para **dibujar en 2D**. Así, podemos representar nuestros propios gráficos sobre un Canvas (también llamado lienzo) o bien modificar una View (en español vista) para personalizar su aspecto, o lo que en habla inglesa se conoce como "look and feel". Cuando nos planteamos dibujar en 2D tenemos dos caminos:

angellea (glitterbug)

- **Dibujar los gráficos o animaciones en un objeto View de nuestro layout.** De esta manera, el dibujo de nuestro gráficos se gestiona por el proceso de dibujado que rige en la jerarquía de la vista, es decir, nosotros simplemente definimos los gráficos que van dentro de la vista.
  Es la mejor si lo que queremos es dibujar gráficos sencillos que no necesitan cambiar dinámicamente y no forman parte de un juego que requiera alto desempeño. Así, por ejemplo, deberíamos dibujar gráficos en una vista si queremos mostrar un gráfico estático o animación predefinida, etc.
- **Dibujar nuestros gráficos directamente en un Canvas.** De este modo, nosotros llamamos personalmente al método **onDraw()** apropiado de la clase, pasándole el **Canvas**, o uno de los métodos **draw...()** del **Canvas**, como **drawPicture()**, etc.
  Es lo mejor si nuestra aplicación necesita redibujarse regularmente . Por eso, aplicaciones como los videojuegos deberían dibujar sobre Canvas.

Autoevaluación

En el caso que una aplicación tenga que dibujar regularmente su vista es recomendable dibujar los gráficos en un objeto View dentro del layout.

○ Verdadero ○ Falso

1.2.- La clase Canvas

La clase Canvas representa un lienzo donde podemos dibujar. Esta clase proporciona una serie de métodos que permiten dibujar líneas, círculos, rectángulos, texto, etc. Lo que hay que dibujar en el lienzo se debe escribir en el método onDraw() de forma que sobrescribiremos este método para crear nuestra interfaz de usuario personalizada. Finalmente para dibujar en un Canvas necesitamos definir un pincel mediante la clase Paint, donde indicaremos el color, grosor de trazo, transparencia, y otras propiedades.

Gerry Dulay

Las vistas se suelen dibujar con frecuencia, con lo cual se ejecuta el método onDraw() varias veces. Inicializar los componentes en este método supone un coste de rendimiento. con lo que crearemos un método llamado init() que llamaremos desde el constructor de la clase Canvas. Puedes consultar la documentación oficial Cómo crear objetos de diseño para ver un ejemplo.

Podemos resumir que con la clase Canvas indicamos *qué* diseñar y con la clase Paint *cómo*.

En la siguiente tabla te presentamos algunos de los métodos de la clase Canvas. No es un listado completo, por lo tanto es recomendable que consultes la documentación en Android Developers sobre la API del objeto Canvas.

Métodos de la clase Canvas.

Función	Métodos
Dibujar figuras geométricas	drawCircle(float cx, float cy, float radio, Paint paint) drawOval(RectF rect, Paint paint) drawRect(RectF rect, Paint paint) drawPoint(float x, float y, Paint paint) drawPoints(float[] pts, Paint paint)
Dibujar líneas y arcos	drawLine(float iniX, float iniY, float finX, float finY,Paint paint) **drawLines(float[] pts, Paint paint)** **drawArc(RectF rect, float startAngle, float angle,boolean useCenter,Paint paint)** **drawPath(Path trazo, Paint paint)**
Dibujar texto	drawText(String text, float x, float y, Paint paint) **drawTextOnPath(String text, Path path, float hOffset, float vOffset, Paint paint)** drawPosText(String text, float[] position, Paint paint)
Dibujar imágenes	drawBitmap(Bitmap bitmap, Matrix matrix, Paint paint)<code>
Tamaño del Canvas	int getHeight()<code> int getWidth()<code>

A la hora de dibujar podemos modificar el área donde dibujar:

- Reducir la región de la pantalla en la que realizaremos las operaciones de dibujo pueden escribir. Podemos establecer un área rectangular mediante Canvas.clipRect(float left, float top, float right, float bottom). No tiene por qué ser rectangular mediante Canvas.clipPath(Path path) o Canvas.clipRegion(Region region)
- También podemos definir una matriz de 3x3 mediante Canvas.setMatrix(matrix) que nos permitirá transformar coordenadas aplicando una traslación, escala o rotación:
  - concat(Matrix matriz)
  - translate(float dx, float dy)
  - scale(float sx, float sy)
  - rotate(float degrees, float px, float py)

- skew(float dx, float dy)

Para saber más

Te proponemos el siguiente taller de Android Developer donde aprenderás a dibujar múltiples formas en un lienzo creando diferentes regiones de recorte:

Advanced Android 11.1 Part C: Apply clipping to Canvas objects

Autoevaluación

Para dibujar figuras geométricas, emplearemos el método:

- ○ drawCircle()
- ○ drawRect()
- ○ drawOval()
- ○ Todos los anteriores son métodos válidos.

1.3.- La clase Paint

La mayor parte de los métodos de la clase Canvas utilizan un parámetro de tipo Paint. Con esta clase podrás definir el color, estilo y el grosor del trazado de un gráfico vectorial.

Coyot (Licencia Pixabay)

Para indicar los colores, en Android se usan enteros de 32 bits, divididos en 4 campos de 8 bits: alfa, rojo, verde y azul (o ARGB por las iniciales en inglés). Cada campo, al estar formado por 8 bits, puede tomar $2^8 = 256$ valores diferentes.

Los componentes rojo, verde y azul se usan para definir el color y el componente alfa define el grado de transparencia con respecto al fondo. Un valor 255 representa un color opaco y a medida que se va reduciendo el valor el dibujo se irá haciendo transparente.

Hay un detalle importante que tenemos que tener en cuenta, y que probablemente habrás visto en otros módulos del ciclo, y es referente a la conveniencia de separar el diseño de la programación en sí. En este sentido, cuando definimos un color tenemos varias opciones para hacerlo:

int miColor;

- **miColor = Color.RED ; // Rojo**
- **miColor = Color.argb(127, 0, 255, 0) ; // Verde transparente**
- **miColor = 0x7F00FF00 ; // Verde transparente**
- **miColor = getResources().getColor(R.color.color_Triangulo);**

Pero, ¿cuál es la mejor opción?

En la línea de lo que decíamos, se recomienda utilizar la última opción listada para no definir los colores directamente en código, sino utilizar el fichero de recursos res/values/colors.xml.

Para saber más

En el enlace que hay a continuación, puedes ver el código del color que selecciones en la paleta que aparece en la parte superior, en el centro.

Acceder a paleta de colores

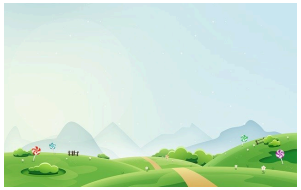
Ejercicio Resuelto

Ahora, puedes descargar un proyecto muy sencillo donde se muestra un ejemplo. En él se crea una vista dibujada por código por medio de un Canvas. Enumeramos los pasos que hemos realizado para que puedas crear el ejemplo:

Creamos un proyecto y seleccionamos Empty Activity.
Creamos una clase personalizada DrawView que hereda de View y sobrescribiremos su método onDraw() donde dibujaremos un rectángulo verde y un círculo amarillo.
Finalmente, le diremos a la Activity que nuestro objeto View será su vista mediante el método setContentView(new RectView(this)).

Mostrar retroalimentación

1.4.- La clase Path



seventhemes (Licencia Pixabay)

Con esta clase puedes definir un trazado a partir de segmentos de línea y curvas. Se puede por tanto definir caminos geométricos que consisten en segmentos de líneas rectas, curvas, curvas cuadráticas y cúbicas. Así, el trazado puedes dibujarlo con canvas.drawPath(Path, Paint).

Con la clase Path también puedes dibujar un texto sobre el trazado determinado. Prueba a poner entre comentarios o bien a eliminar el código que hemos probado en el método onDraw() del ejercicio resuelto anterior, y cámbialo por éste que ves a continuación. Cambia más cosas y experimenta.

```
…
// Crear un trazo
   Path trazo = new Path();
// Definir un círculo
   trazo.addCircle(160, 160, 120, Direction.CCW);
   canvas.drawColor(Color.GRAY);
// Crear un pincel y definirle las características
   Paint pincel = new Paint();
   pincel.setColor(Color.RED);
   pincel.setStrokeWidth(5);
   pincel.setStyle(Style.FILL_AND_STROKE) ;
// Dibujar el trazo sobre el canvas con los datos
// definidos para el pincel
   canvas.drawPath(trazo, pincel);
// Características del pincel para escribir texto
   pincel.setStrokeWidth(1);
   pincel.setStyle(Style.FILL);
   pincel.setTextSize(20);
   pincel.setTypeface(Typeface.MONOSPACE);
// Escribir el texto
   canvas.drawTextOnPath("DAM y DAW. en IES. Aguadulce",
                  trazo, 10, 40, pincel);
…
```

Debes conocer

Tienes en el siguiente vídeo información sobre los métodos de la clase Paint y **Path.**

La clase Paint en Android

Resumen textual alternativo

Autoevaluación

Señala qué método corresponde a la clase Path:

- ○ setColorBW(int colorbw)
- ○ drawCircle(float x, float y, float radio, Paint painty)
- ○ moveTo(int x, int y)
- ○ rotate(float grades, float xCenter, float yCenter)

1.5. Aplicando movimiento a los objetos

Hay dos formas básicas de dibujar en 2D en Android:

1. **Dibujar nuestros gráficos y animaciones dentro de un View**: de esta forma el sistema se encarga de dibujar y animar por nosotros. Esta opción es la recomendada cuando se trata de objetos estáticos o animaciones que no requieren de mucho procesamiento. Hay ciertos juegos como el ajedrez, mahjong, sudoku que no requieren mucho trabajo en la parte gráfica.
2. **Dibujar dentro de un Canvas**: si el objeto está en movimiento, o necesita ser redibujado con frecuencia, entonces es mejor usar un Canvas donde el desarrollador toma el control de lo que se dibuja sobre la pantalla. Si optamos por esta opción tenemos dos formas de realizar el movimiento de un objeto:
   - Sobre el hilo principal de la aplicación (UI Thread). Sólo escogeremos esta opción cuando las operaciones de dibujo y movimiento no consuma demasiado tiempo en ejecutarse como es el caso del ejemplo que te proponemos en este apartado.
   - En un hilo aparte que dibuje sobre un Canvas tan rápido como nos sea posible.

Steve Webster.

Recomendación

Te recomendamos que estudies el código del ejemplo que te proporcionamos en el siguiente enlace para su descarga, en el que se presenta un círculo que se mueve por pantalla.

Círculo moviéndose por la pantalla. (zip - 12.45 MB)

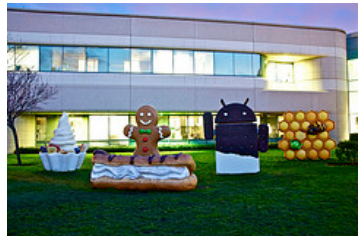
Autoevaluación

Cuando un objeto se redibuje con bastante asiduidad es mejor utilizar un View.

○ Verdadero ○ Falso

1.5.1.- La clase SurfaceView

¿Qué nos aporta la clase SurfaceView?



LuisJouJR.

Es una subclase especial de View, que ofrece una superficie de dibujo dedicada dentro de la jerarquía View. El objetivo de esta clase es ofrecer la superficie de dibujo a un hilo secundario de la aplicación, para que ésta no tenga que esperar a que la jerarquía View del sistema esté lista para dibujar. Así, en lugar de tener que esperar, un subproceso secundario que tiene referencia a una SurfaceView puede dibujar en su Canvas a su propio ritmo.

Para empezar, es necesario crear una nueva clase que hereda de SurfaceView. La clase debe implementar también SurfaceHolder.Callback. Esta subclase es una interfaz que notificará la información sobre la superficie subyacente, por ejemplo, cuando se crea, cambia o es destruida. Estos eventos son importantes para que sepas cuándo se puede empezar a dibujar, si es necesario hacer ajustes basados en las nuevas propiedades de la superficie, y cuándo parar de dibujar.

Vamos a crear un proyecto y vamos a realizar los siguientes pasos:

1. Creamos una clase que hereda de SurfaceView que será la clase encargada de gestionar el hilo secundario y recibir las entradas del usuario y enviar el evento al hilo secundario.
2. Cuando otros objetos quieran gestionar el objeto de superficie o SurfaceView no lo harán directamente sino que lo harán a través de la interfaz SurfaceHolder que implementa nuestra clase SurfaceView. Es probable que se pueda tener más de un objeto de superficie pero la forma de trabajar con las clases SurfaceView es estándar utilizando esta interfaz. Por lo tanto, cuando se inicia nuestro SurfaceView tenemos que:
   - Crear una variable SurfaceHolder e inicializarla llamando al método getHolder().
   - A continuación, deberíamos notificar a **SurfaceHolder** que queremos recibir devoluciones de llamada SurfaceHolder (desde SurfaceHolder.Callback) llamando addCallback().
   - Finalmente reemplazaremos cada uno de los métodos SurfaceHolder.Callback dentro de nuestra clase SurfaceView.

```
public class GameSurfaceView extends SurfaceView implements SurfaceHolder.Callback {
   // Variables globales<br />    <strong>private </strong>SurfaceHolder surfaceHolder = null;
   ...
   // Constructor de la clase GameSurfaceView
   public GameSurfaceView (Context context, AttributeSet attrs) {
      super(context, attrs);<br />        <strong>surfaceHolder=getHolder();</strong>
<strong>       surfaceHolder.addCallback(this);</strong>
      ...
   }

   public void surfaceCreated(SurfaceHolder holder) {
      ...
   }

   public void surfaceDestroyed(SurfaceHolder holder) {
      ...
   }

   public void surfaceChanged(SurfaceHolder holder, int format, int width,  int height) {
      ...
   }

}
```

3. Añadimos la clase creada SurfaceView al archivo activity_main.xml y un texto que aparecerá en la parte inferior.

```
<com.example.surfaceview.GameSurfaceView
    android:id="@+id/gameSurfaceView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
```

```
app:layout_constraintBottom_toTopOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent" />
```

Debes conocer

En el siguiente enlace se puede ver un ejemplo de cómo dibujar en la clase View y cómo hacerlo usando la clase SurfaceView.

Dibujar gráficos en Android

Y este otro, puedes consultar la documentación oficial de la clase SurfaceView en Android.

Usando SurfaceView en Android

1.5.2.- Implementación del hilo secundario

Myriams-Fotos (Licencia Pixabay)

Crearemos el hilo secundario o clase Thread dentro de la clase SurfaceView. Esta clase llevará a cabo todos los procedimientos de dibujo al Canvas. Si estamos desarrollando un juego, esta clase es la que tiene la mayor carga de trabajo, sería nuestro motor de juego (***Game Engine***) ya que sería la responsable de:

Manejar la lógica del juego (cargar recursos, mover personajes, controlar las colisiones etc).
Procesar las entradas del usuario que recibe de la clase SurfaceView
Dibujar en la pantalla y actualizar los componentes.

Siguiendo con el ejemplo anterior vamos a crear un juego que pinte un círculo en la pantalla de la superficie en el lugar donde hemos pulsado en la pantalla. Para ello debemos realizar los siguientes pasos:

Crear una clase interna que hereda de Thread, en nuestro caso la llamaremos GameThread. La ventaja de ser una clase interna es que puede acceder a los atributos de la clase SurfaceView simplificando el código, ya que no es necesario pasar en el constructor de la clase GameThread el objeto SurfaceHolder.
Crear un método update() que servirá para la ejecución principal del juego. En nuestro ejemplo no tenemos que controlar nada, con lo cual será un método vacío.
Crear un método que dibuje sobre la pantalla, lo llamaremos doDraw(). Este método es el encargado de dibujar el círculo en el punto (X,Y) del evento MotionEvent.
Sobrescribir el método run() que:
Obtendrá un objeto Canvas a través del objeto SurfaceHolder del constructor surfaceHolder.lockCanvas(null).
Actualizar la lógica del juego llamando al método update().
Dibujar sobre el objeto Canvas llamando al método doDraw().
Ahora hay que decirle a la pantalla principal que debe mostrar lo que se dibujó sobre el Canvas. Esto se realiza mediante la siguiente llamada: surfaceholder.unlockCanvasAndPost(canvas).

```
class GameThread extends Thread {
    //Flag que usamos para indicar si el hilo debe correr o no.
    private boolean flagRun = false;
    private Canvas canvas = null;
    @Override
    public void run() {
      //empieza el ciclo principal del juego
      while (flagRun) {
        try {
          //obtenemos una instancia al canvas de la superficie
<strong>            canvas = surfaceHolder.lockCanvas(null);</strong>
          //nos aseguramos de que solo un hilo a la vez manipule la superficie
          synchronized (surfaceHolder) {
            update();//actualizamos la lógica del juego
            doDraw();//dibujamos sobre el canvas
          }
        } finally {
          if (canvas != null) {
<strong>              surfaceHolder.unlockCanvasAndPost(canvas);</strong>
          }
        }
      }
    }
    private void update() {
      //actualiza la lógica aquí
    }
    private void doDraw() {
      drawBall();
```

```
    }
    public void setRunning(boolean b) {
        flagRun = b;
    }
    /* This method will be invoked to draw a circle in canvas. */
    public void drawBall() {
        // Get and lock canvas object from surfaceHolder.
        Paint surfaceBackground = new Paint();
        // Set the surfaceview background color.
        surfaceBackground.setColor(Color.CYAN);
        // Dibujamos una superficide de un color
       canvas.drawRect(0, 0, GameSurfaceView.this.getWidth(), GameSurfaceView.this.getHeight(), surfaceBackground);
        // Dibujamos el círculo en las coordenadas iniciales
        canvas.drawCircle(circleX, circleY, 100, paint);
    }
}
```

Puedes descargarte el código completo del proyecto en el siguiente enlace:

Código del ejemplo completo GameSurfaceView (zip - 0.37 MB) .

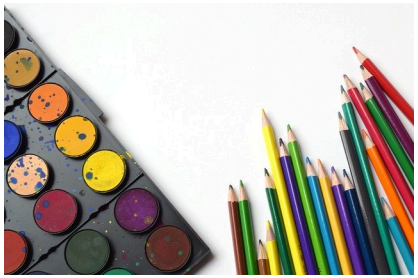
Para saber más

Acerca de los hilos o threads, puedes ampliar más la información en el siguiente enlace:

Hilos en Android.

1.6.- La clase Drawable

La documentación de Google nos dice que **Drawable** es una abstracción general de "algo que puede ser dibujado".



Mimzy (Licencia Pixabay)

A partir de esa definición nosotros podemos definir objetos gráficos más específicos a través que clases que heredan de ella. Puedes consultar la documentación en Recursos de elementos de diseño para ver el conjunto de clases que heredan de Drawable: **BitmapDrawable, ShapeDrawable, VectorDrawable, LayerDrawable, StateListDrawable, GradientDrawable, TransitionDrawable** y AnimationDrawable entre otras.

Nos detendremos un poco con la clase VectorDrawable.

Cuando tenemos que hacer uso de muchos recursos gráficos es conveniente el uso de **Gráficos Vectoriales Escalables** o SVG ya que disminuye el tamaño de nuestro APK. Utilizar este recurso es útil en imágenes que no sean complejas, ni tenga patrones ni degradados; en estos supuestos es mejor utilizar los recursos convencionales de mapa de bits. Si se abre un fichero con extensión .svg en un editor de texto encontraremos el código que indica cómo pintar la imagen de forma que puede ser redimensionada al no estar generada por píxeles. Y para agregar la imagen utilizamos un ImageView mediante el atributo app:srcCompat. Consulta la documentación en Android Developers: Cómo agregar gráficos vectoriales de varias densidades.

Un recurso Drawable se puede definir en código como muestra el siguiente código donde usamos la clase ShapeDrawable para dibujar un rectángulo.

```
public class FormasDibujables extends View {
    ShapeDrawable sdRectan ;
    public FormasDibujables(Context context,
        AttributeSet attrs) {
        super(context, attrs);
        // Crear un rectángulo
        sdRectan = new ShapeDrawable(new RectShape());
        sdRectan.setBounds(200, 5, 300, 100) ;
        // De color verde
        sdRectan.getPaint().setColor(Color.GREEN) ;
    }
    @Override
    protected void onDraw(Canvas canvas) {
        // Pintar el rectángulo
         sdRectan.draw(canvas) ;
    }
}
…
```

Hay tres formas de definir e instanciar un objeto:

- **Usando una imagen guardada** en los recursos del proyecto.
- **Utilizando un fichero XML** que define las propiedades del objeto para luego recuperarlo mediante **getDrawable()** y el ID que se le asigne en el XML.
- **Usando los constructores** propios de la clase.

Debes conocer

Este vídeo sobre la clase Drawable y sus descendientes es fundamental que lo veas por su interés.

Clase Clase Drawable

Resumen textual alternativo

Para saber más

Te ponemos a continuación un enlace donde puedes ver ejemplos en código XML de clases que heredan de Drawable y su uso en el diseño de una interfaz:

Drawables en Desarrollo de Aplicaciones para Dispositivos Móviles.

Autoevaluación

¿Qué subclase de Drawable permite hacer animaciones frame a frame partiendo de objetos Drawable?

- ○ StateListDrawable.
- ○ LayerDrawable.
- ○ TransitionDrawable.
- ○ AnimationDrawable.

1.6.1.- Cargar imágenes

Uno de los usos más básicos a la hora de usar gráficos por pantalla consiste en la carga de imágenes almacenadas en las carpetas de recursos Drawable. De este modo, puedes mostrar una imagen mediante el componente ImageView. Además del identificador o id hay otros atributos que nos ayudan a configurar esta clase en nuestro fichero XML:

- src: para establecer el archivo fuente de la imagen. En el caso que se haga mediante código utilizaremos el método setImageResource().
- background: para establecer un color o un dibujo en el fondo de un ImageView. En código Java equivale al método setBackgroundColor(Color.BLACK).
- padding: para establecer el relleno de la imagen (paddingRight,paddingLeft, paddingTop, paddingBottom o bien padding para todos los lados).
- scaleType: para controlar cómo se debe cambiar el tamaño o mover la imagen para que coincida con el tamaño del componente ImageView. El valor de escalado puede ser:
  - center: muestra la imagen centrada en la vista sin escala.
  - center_crop: ajusta la imagen de manera que las dimensiones x e y sean mayores o iguales que la vista, manteniendo la relación de aspecto de la imagen; centra la imagen en la vista.
  - center_inside: ajusta la imagen para que quepa dentro de la vista, manteniendo la relación de aspecto de la imagen. Si la imagen ya es más pequeña que la vista, entonces esto es lo mismo que centrar.
  - fit_center **(opción por defecto)**: ajusta la imagen para que quepa dentro de la vista, manteniendo la relación de aspecto de la imagen. Al menos un eje coincidirá exactamente con la vista, y el resultado se centra dentro de la vista.
  - fit_start: igual que fitCenter pero alineado a la parte superior izquierda de la vista.
  - fit_end: igual que fitCenter pero alineado en la parte inferior derecha de la vista.
  - fit_xy: permite establecer el tamaño exacto de la imagen en su diseño. Sin embargo, pueden producirse posibles distorsiones de la imagen debido a la escala.

Puedes ver, en el código de la clase que se muestra más abajo, dos métodos para cargar imágenes, en concreto dos imágenes. Ambas fotos deben estar almacenadas en nuestra carpeta de recursos, en res/drawable-mdpi

Con esas imágenes en los recursos, creamos un proyecto y en la clase que se nos crea por defecto, en el método onCreate() introducimos el código que nos interesa. Como puedes observar en el código de la misma, se cargan dos imágenes, cada una cargada de una manera, aunque con idéntico resultado:

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        LinearLayout miLay = new LinearLayout(this) ;
        miLay.setOrientation(LinearLayout.VERTICAL) ;

        // Crear objeto ImageView
        ImageView miImagen = new ImageView(this) ;
        miImagen.setImageResource(R.drawable.ic_launcher) ;

        // Crear objeto Drawable
        Drawable miObjetoDrawable = this.getResources().getDrawable(R.drawable.vigilante) ;
        // Crear objeto ImageView
        ImageView miSegundaImg = new ImageView(this) ;
        miSegundaImg.setImageDrawable(miObjetoDrawable) ;

        // añadir al layout las imágenes
        miLay.addView(miImagen) ;
        miLay.addView(miSegundaImg) ;

        setContentView(miLay) ;
    }
}
```

Código de la clase para cargar imágenes. (1 KB)

Citas para pensar

Podrán cortar todas las flores, pero no podrán detener la primavera.

Pablo Neruda

Autoevaluación

Las imágenes que usemos como recursos en nuestra aplicación, normalmente estarán ubicadas en la carpeta:

- ○ res/imageview
- ○ res/onCreate
- ○ res/fotos
- ○ res/drawable-mdpi

1.6.2.- Efectos: transiciones

fisserman

La clase TransitionDrawable hereda de LayerDrawable, que es descendiente de Drawable, que permite efectos de transición en una primera y una segunda capa. Para iniciar la transición hay que llamar al método startTransition(int tiempo). Sólo permite la transición entre dos capas, se puede volver a la primera capa con el método resetTransition().

A continuación tienes un ejemplo que puedes descargar, de un proyecto donde hemos seguido los siguientes pasos para su construcción:

Creamos un nuevo proyecto con nombre ***Efectos***.
Crea el fichero de recursos en **res/drawable/transicion.xml** con el código que puedes ver en el proyecto, donde ponemos los dos elementos primera y segunda.
Copiamos dos fotos con esos nombres, ***primera*** y ***segunda*** en la carpeta de recursos.
En el fuente, TransicionActivity.java lo hemos llamado en este caso, escribimos el código que ves en dicho proyecto y que básicamente instancia un objeto de la clase **TransitionDrawable** e inicia la transición. También, se añade un oyente para iniciar la transición al hacer clic en la imagen.

Igualmente una transición se puede definir en un fichero XML. Mediante el elemento transition cada objeto Drawable en la transición se define en un item anidado. Puedes consultar la documentación oficial Definir un elemento de diseño de transición y realizar el mismo ejercicio que te hemos planteado en XML.

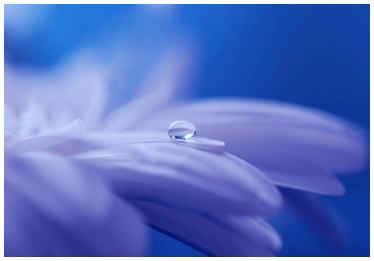
Recomendación

Te recomendamos que descargues el siguiente proyecto, lo importes como proyecto de código existente, lo ejecutes y estudies el código detenidamente.

Efecto de transición (zip - 12.79 MB)

1.6.3.- Fondos

Con la clase Drawable podemos **crear fondos** para usarlos **en botones** o **como fondo de pantalla**.



DreamyArt (Licencia Pixabay)

Mediante el uso de los recursos XML, puedes crear fondos personalizados con diferentes formas y no preocuparte sobre los tamaños de pantalla. Puedes definir diferentes formas de fondo como oval, anillo, un rectángulo o una línea, aplicar esquinas redondeadas para formas rectangulares, sólidos, degradados, etc.

En el proyecto que puedes descargarte más abajo, ponemos un fondo azul con esquinas redondeadas. Los pasos seguidos son bien sencillos:

Crear el proyecto y crear un fichero **.xml** llamado **rectangulo.xml** en la carpeta **res/drawable-mdpi** con el sencillo código que ves en él.
Establecer en el fichero principal del layout, en nuestro caso **activity_main.xml**, el fondo que vas a usar mediante **android:background="@drawable/rectangulo"**.
En **strings.xml** hemos puesto el mensaje que deseábamos mostrar.

Poner fondo azul. (zip - 18.67 MB) (13.5 MB)

Para saber más

En el siguiente enlace puedes profundizar sobre la mejora en el diseño del aspecto de las aplicaciones, con la siguiente herramienta:

La herramienta draw9patch del SDK de Android

Autoevaluación

Con Android, podemos crear fondos con formas:

- [ ] De anillo.
- [ ] Ovales.
- [ ] Rectangulares, pero sin poder aplicar esquinas redondeadas.
- [ ] Lineales.

Mostrar retroalimentación

1.6.4.- Animaciones

kalleboo.

En Android encontramos todo un mundo al hablar de animaciones. Principalmente podemos utilizar cuatro tipos de animaciones:

- **Animaciones de marcos o Drawable:** este tipo de animaciones son las más sencillas y se basa en la sucesión de varios Drawable a intervalos de tiempos específicos. Para crear esta animación frame a frame se utiliza la clase AnimationDrawable. En XML se utiliza la etiqueta animation-list que funciona como un recurso Drawable.
- **Animaciones de layouts:** este tipo de animaciones se utilizan para animar la transición entre layouts y entre actividades. Se puede seleccionar si se quiere cambiar toda la interfaz o sólo una vista de la interfaz. Este tipo de animaciones sólo necesita el diseño inicial y el diseño final para realizar la transición. Consulta el enlace Anima los cambios de diseño con una transición de la documentación oficial.
- **Animaciones de vistas:** también conocidas por **Tween** que permiten crear crear efectos de rotación, traslación, zoom y transparencia en cualquier vista. La animación se calcula entre el punto inicial y final de un objeto View y se suele definir en XML utilizando como elemento raíz una etiqueta según tipo de animación en base a su posición <translate>, tamaño <scale>. rotación <rotate> o transparencia <alpha>. Si la animación consta de un grupo de elementos se debe utilizar <set>; como puedes ver en el ejemplo Animación de vistas de la documentación oficial.
- Animación de propiedades: estas animaciones se basan en animar las propiedades de un objeto. Este tipo de animaciones son más completas que las animaciones de vistas que se basan en el cambio de tamaño o rotación ya que se puede especificar qué propiedades se modificarán en el tiempo, establecer un conjunto de animaciones que se reproducen juntas, la frecuencia con la que queremos actualizar cada frame, etc. Puedes consultar el siguiente enlace que describe la **animación de propiedades**. Se suele utilizar los elementos raíz <objectAnimator>, <valueAnimator> y <set> para agruparlos cuando se define este tipo de animaciones en un XML.

Te recomendamos a continuación que veas un ejemplo sencillo sobre una animación frame a frame:

Creamos un fichero XML donde definimos la animación de imágenes que tenemos en la carpeta Drawable y establecemos el tiempo en milisegundos para mostrar este marco.

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android" >
   <item android:drawable="@drawable/s1" android:duration="50" />
   <item android:drawable="@drawable/s2" android:duration="50" />
   <item android:drawable="@drawable/s3" android:duration="50" />
   <item android:drawable="@drawable/s4" android:duration="50" />
   <item android:drawable="@drawable/s5" android:duration="50" />
</animation-list>
```

A continuación en código cargamos el objeto ImageView de nuestra interfaz que alojará la animación y establecemos que su fondo será nuestro recurso XML AnimationDrawable.
Obtenemos la animación que se ha compilado en un objeto AnimationDrawable.
Y finalmente iniciamos la animación que por defecto se reproduce en bucle. En nuestro ejemplo la animación se ejecuta cuando se pulsa un botón es por ello que comprobamos si la animación está ejecutándose la pararíamos en caso contrario se inicia.

```
ImageView imgView = (ImageView)findViewById(R.id.imagen);
imgView.setVisibility(ImageView.VISIBLE);
imgView.setBackgroundResource(R.drawable.frame_animation);

AnimationDrawable frame = (AnimationDrawable) imgView.getBackground();
if(frame.isRunning()){
      frame.stop();
}else{
      frame.start();
}
```

Recomendación

Sencilla animación realizada con unas imágenes tomadas de la hoja de sprites sobre el personaje **Príncipe de Persia**, realizada por Broderbund.

Ejemplo de animación por marco (zip - 12.79 MB)

Para saber más

Las animaciones han ido cambiando a medida que ha evolucionando Android, ya que la forma de interactuar con el usuario se traduce en éxito y calidad en nuestras aplicaciones. Hemos señalado cuatro tipo de animaciones más habituales y que nosotros controlamos, pero es posible utilizar las animaciones en base al mundo real a través de la física.

Si quieres saber más, consulta la documentación oficial:

Introducción a las animaciones en Android.

1.7.- Gráficos 3D

Android incluye soporte para gráficos mediante la API OpenGL, concretamente OpenGL ES.

OpenGL ES es un aderezo a la especificación OpenGL pensada para dispositivos embebidos.

La API específica proporcionada por Android es similar a la API J2ME JSR239 de OpenGL ES, aunque no idéntica.

La creación de aplicaciones 3D usando OpenGL está más allá del alcance de este módulo y si te interesa el tema, se te deja como investigación, pero básicamente, los pasos a seguir para utilizar esta API, de modo muy resumido serían:

- Escribir una subclase **View** personalizada.
- Obtener un identificador para un **OpenGL Context**, que proporciona acceso a la funcionalidad de OpenGL.
- En el método **OnDraw()** de tu vista, obtener un identificador para un objeto **GL**, y utilizar sus métodos para realizar las operaciones.

PIRO4D (Licencia Pixabay)

Para ver un ejemplo de este modelo de uso, que muestra cómo utilizar con hilos, puedes mirar este fuente de los ejemplos que vienen con el SDK:

Ejemplo clase para 3D (java - 1.97 KB)

Para saber más

En este enlace puedes ver esquemáticamente algo más sobre gráficos en Android.

Gráficos 2D y 3D con Android.

Autoevaluación

OpenGL ES es un añadido a la especificación OpenGL que surge para dispositivos embebidos.

○ Verdadero ○ Falso

2.- Reproducción multimedia

Caso práctico

María y **Juan** han decidido repartirse parte del trabajo para luego poner en común lo que hayan aprendido. Han visto que a la hora de manipular y presentar los contenidos multimedia hay que tener claro y comprender el ciclo de vida de una actividad en Android que ya estudiaron anteriormente. También deben repasar qué formatos soporta y las clases disponibles en Android.

María se ha puesto a trabajar con los reproductores mientras que **Juan** lo va a hacer con las capturas de audio y vídeo. Posteriormente, en una reunión con **Ada** y el resto del equipo pondrán en común todas las conclusiones a las que hayan llegado. Hoy en día, uno de los usos más comunes de los teléfonos móviles es su utilización como reproductor multimedia: reproductor MP3, para la reproducción de vídeos a través de Internet, etc. Android presenta muchas facilidades para la reproducción multimedia y así, permite la reproducción de un montón de formatos, tanto de audio como de vídeo.

Una cosa que tienen todos clara es el ciclo de vida de una aplicación, o sea, cómo las aplicaciones se crean, se ponen en espera y finalmente se destruyen. Además lo que más les llama la atención es que el ciclo de vida de una aplicación Android es distinto al de una aplicación en otro sistema como pueda ser Windows. Principalmente porque en Android el ciclo de vida es controlado por el sistema, en lugar de ser controlado directamente por el usuario.

Una aplicación en Android está formada por un conjunto de elementos básicos de visualización: **las actividades**.

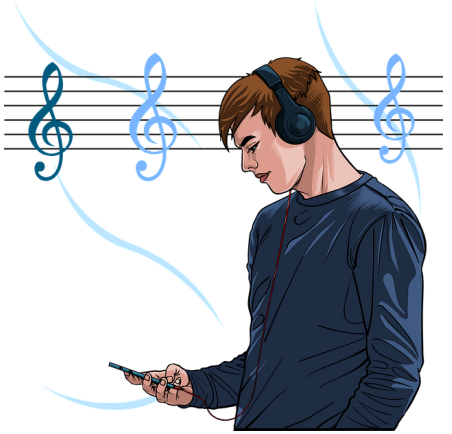
Además de varias actividades, una aplicación también puede contener **servicios**.

Las actividades controlan el ciclo de vida de las aplicaciones, puesto que el usuario no cambia de aplicación, sino de actividad. El sistema guarda una pila con las actividades previamente visualizadas, así el usuario puede regresar a la actividad anterior pulsando la tecla atrás.

En Android, el sistema determina cuándo destruir el proceso, teniendo en cuenta las partes de la aplicación que están ejecutándose (actividades y servicios), la prioridad que tengan para el usuario y cuánta memoria disponible hay en un determinado momento.

Si tras eliminar el proceso de una aplicación, el usuario vuelve a ella, se crea de nuevo el proceso, pero se habrá perdido el estado que tenía esta aplicación. En estos casos, va a ser responsabilidad del programador almacenar el estado de las actividades, si queremos que cuando sea reiniciada conserve su estado. Recuerda los estados por los que toda actividad puede pasar: **activa, pausada y parada**.

Los recursos multimedia son una de las funcionalidades más utilizadas en los dispositivos móviles. Android ha ido mejorando esta característica en cada nueva versión proporcionando una colección de API que permite manejar los recursos multimedia de forma muy sencilla.



Chakkree Chantakad (Licencia Pixabay)

2.1.- Clase Mediaplayer

La clase MediaPlayer se puede utilizar para gestionar la r**eproducción de los archivos y secuencias de audio y vídeo**.

dogulove

Un objeto MediaPlayer puede pasar por diversos estados:

- Inicializados sus recursos ***(initialized).***
- Preparando la reproducción ***(preparing).***
- Preparado para reproducir ***(prepared).***
- Reproduciendo (started), en pausa ***(paused).***
- Parado ***(stoped).***
- Reproducción completada (***playback completed***).
- Finalizado (***end***) y con error ***(error).***

Es interesante saber el estado en que se encuentra, puesto que muchos métodos sólo pueden ser llamados desde ciertos estados. Así, no podemos ponerlo en reproducción (start()) si no está en estado preparado. Si se hace una llamada a un método no admitido, para un determinado estado, se producirá un error de ejecución.

Para saber más

Puedes consultar el diagrama de estados en el siguiente enlace:

Diagrama de estados.

2.2.- Reproducir sonidos y vídeos

¿Se puede incluir el audio o el vídeo en nuestra aplicación como un recurso?

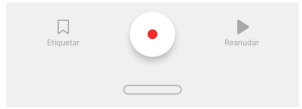
Por supuesto que sí. Simplemente, tendrías que seguir los siguientes pasos:

- Crear una nueva carpeta con nombre **raw** dentro de la carpeta **res**.
- Copiar a esa carpeta el fichero de audio o vídeo, como por ejemplo **xanadu.mp3.**
- Añadir el siguiente código:


```
MediaPlayer mep = MediaPlayer.create(this, R.raw.xanadu);
mep.start() ;
```

Para parar la reproducción tendrías que usar el método stop(). Y si después, quieres volver a reproducirlo, deberías usar el método prepare() y a continuación start(). También puedes usar pause() y **start()** para ponerlo en pausa y reanudarlo.

Otra opción, en vez de meterlo como recurso en el proyecto, es reproducirlo desde un fichero utilizando el código de más abajo. En este caso hay que llamar al método prepare(). En el caso anterior no era preciso porque esa llamada se hace desde create().



Captura del IDE Android Studio. Apache 2.0

```
MediaPlayer mep = new MediaPlayer();
mep.setDataSource(Ruta_al_fichero_a_Reproducir);
mep.prepare();
mep.start();
```

Para saber más

Para reproducir sonido en segundo plano, como un servicio, puedes consultar este interesante tutorial:

Reproducción de sonido de fondo.

Otro tutorial también bastante recomendable, sobre la reproducción de sonidos, es el siguiente:

Reproducir sonidos.

Página donde se muestra paso a paso la inclusión de vídeos en una aplicación Android:

Incluir vídeo.

Autoevaluación

Para poner en pausa un sonido usaremos el método pause().

○ Verdadero ○ Falso

3.- Aplicaciones multimedia

Caso práctico

Ada quiere hacer un pequeño encargo a **María**. Se trata de que realice un pequeño programita para captura de audio y vídeo en Android.

—Hola, **María** —dice **Ada—**. Hoy tengo una tarea especial para ti.

—¿Sí? ¿De qué se trata? —responde **María**.

—Bueno, quizás esto sea un poco complicado para tí, se trata de captura de audio y vídeo, ¿cómo andas en eso?

—¿Qué? —dice **María** con cierta perplejidad y un tanto preocupada.

—Me alegra que te guste —dice **Ada** esbozando una sonrisa malévola.

Con Android podemos capturar contenido multimedia de varias formas. Probablemente, lo más fácil sea utilizar las aplicaciones que incorpora de serie el terminal Android que vayamos a manejar, pero en cualquier caso, si queremos, podemos programarlo con el uso de tres clases:

Luinfana (GNU/GPL)

- **android.hardware.Camera**: para la captura de imágenes y vídeo.
- **android.view.SurfaceView**: para renderizar las imágenes capturadas por la cámara.
- **android.media.MediaRecorder**: para capturar audio y vídeo.

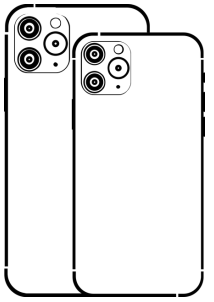
Para saber más

Aquí puedes encontrar más información sobre la gestión de imágenes en Android:

Trabajando con imágenes

3.1.- Captura de imagen y vídeo

La mayoría de dispositivos Android tienen al menos una aplicación de cámara instalada. Para advertir que tu aplicación depende de que tenga cámara, debes poner una etiqueta, un tag en tu fichero ***manifest***:



mayurbhutada (Licencia Pixabay)

```
<manifest ... >
    <uses-feature android:name="android.hardware.camera" />
</manifest ... >
```

Si la aplicación utiliza la cámara, pero no es imprescindible, se aconseja añadir android:required="false" al tag. De este modo, Google Play permitirá a los terminales sin cámara descargar tu aplicación. Es por tanto responsabilidad del programador de la aplicación comprobar la disponibilidad de la cámara en tiempo de ejecución llamando a hasSystemFeature(PackageManager.FEATURE_CAMERA).

En Android, el modo de delegar acciones a otras aplicaciones es invocar una intención (***Intent***) que describe lo que quieres hacer. Este proceso conlleva tres cosas:

- La intención ***Intent*** en sí misma.
- Una llamada para iniciar la ***Activity*** externa.
- Algún código para gestionar los **datos de la imagen** cuando el foco retorne a la aplicación.

Una función que invoca una **intent** para capturar una foto puede ser:

```
private void dispatchTakePictureIntent(int actionCode) {
 Intent takePictureIntent =
       new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
 startActivityForResult(takePictureIntent, actionCode);
}
```

Para comprobar si una aplicación puede gestionar una **intent**, puedes usar un método como éste:

```
public static boolean isIntentAvailable(Context context,
      String action) {
 final PackageManager packageManager =
       context.getPackageManager();
 final Intent intent = new Intent(action);
 List list = pacackageManager.queryIntentActivities(intent,
       PackageManager.MATCH_DEFAULT_ONLY);
 return list.size() > 0;
}
```

Debes conocer

En la siguiente guía se explica detalladamente cómo capturar vídeo y reproducirlo. Para la captura de vídeo se usa la clase MediaRecorder. Para la reproducción se utiliza la clase MediaPlayer. También se usa la clase SurfaceView para la vista previa del vídeo y su reproducción.

Capturando vídeo

Para saber más

En el siguiente proyecto puedes ver una aplicación completa para grabar imagen y vídeo. Te recomendamos que la descargues y estudies el código detenidamente.

Proyecto para capturar imagen y vídeo. (zip - 12.47 MB)

Autoevaluación

No es cosa del programador comprobar la disponibilidad de la cámara en tiempo de ejecución.

○ Verdadero ○ Falso

3.2.- Captura de audio

Podemos capturar audio de una manera muy sencilla, tal y como puedes ver en el siguiente código, y que en resumen consiste en las tres cosas que hacíamos en el apartado anterior:

- **Crear la intención.**
- **Hacer una llamada para iniciar la Activity.**
- **Algún código para gestionar los datos del sonido cuando el foco retorne a la aplicación.**

OpenClipart-Vectors (Licencia Pixabay)

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate (savedInstanceState);
  setContentView(R.layout.activity_sonido_captura) ;
  Intent intencion = new Intent(
    MediaStore.Audio.Media.RECORD_SOUND_ACTION);
  startActivityForResult(intencion, R_GRABADOR) ;
}


@Override
protected void onActivityResult(int idPeticion,
 int resultCode, Intent data) {
  if (idPeticion == R_GRABADOR && resultCode == RESULT_OK) {
    Uri audioFileUri = data.getData() ;
    mPlayer = MediaPlayer.create(this, audioFileUri);
    mPlayer.start() ;
  }
}
```

Recomendación

Puedes probar el proyecto con el código completo de la aplicación, descargándotela de aquí:

Ejemplo de captura de audio. (zip - 12.62 MB)

Debes conocer

En la siguiente página, en inglés, hay un ejemplo comentado, donde se captura audio con la clase MediaRecorder. Puedes ayudarte de un traductor como el de Google para traducir el texto del artículo. Es posible también descargar el código del proyecto.

Ejemplo de captura de audio.

Condiciones y términos de uso de los materiales

Historial de actualizaciones

Versión: 03.00.01	**Fecha de actualización: 03/01/24**	
Actualización de materiales y correcciones menores.		
Versión: 03.00.00	**Fecha de actualización: 14/06/20**	**Autoría: Lourdes Rodríguez Morón**
Ubicación: 1.6.4 **Mejora (tipo 3):** El apartado de animaciones no está correctamente explicado, el enlace ya no está disponible y el ejemplo compila pero no funciona en la versión actual de android **Ubicación:** 1.5 **Mejora (tipo 2):** El ejemplo en el que se basa este apartado no funciona. **Ubicación:** Nodos del mapa conceptual **Mejora (Mapa conceptual):** Actualización del mapa en base a los nuevos contenidos **Ubicación:** Tabla de contenidos **Mejora (Orientaciones del alumnado):** Actualización de la tabla de contenidos en base a los nuevos contenidos		
Versión: 02.00.00	**Fecha de actualización: 25/04/14**	**Autoría: Víctor Gil Rodríguez**
Versión actualizada con Android.		
Versión: 01.00.00	**Fecha de actualización: 25/04/14**	
Versión inicial de los materiales.		