La Clase `Camera`

CONCEPTOS

Antecedentes.

La clase original de clase `Camera`, que ahora es obsoleta, es la siguiente:

```
public class Camera extends Object
java.lang.Object -> android.hardware.Camera
```

Android incluye soporte para varias cámaras y funciones de la cámara disponibles en los dispositivos, lo que permite capturar imágenes y vídeos en las aplicaciones.

Sugerencias.

Antes de activar la aplicación para usar las cámaras en Android, se debe preguntar cómo la aplicación utilizará el hardware.
Requisito de la cámara. ¿Es importante el uso de la cámara para la aplicación y no se desea la instalada en un dispositivo que no tiene una cámara? Si es así, se debe declarar el requisito de la cámara en el manifiesto.
Foto rápida o cámara personalizada. ¿Cómo la aplicación utilizará la cámara? ¿Sólo interesa tomar una imagen o clip rápido de vídeo, o la aplicación será una nueva forma de utilizar la cámara? Para tomar un clip rápido, considerar el uso de las aplicaciones de la cámara existente.
Almacenamiento. ¿Las imágenes o vídeos que la aplicación genera son accesibles solamente a la aplicación o se comparten con otras aplicaciones, como la Galería u otros medios y aplicaciones sociales? ¿Las imágenes y vídeos estarán disponibles incluso si se desinstala la aplicación?

Lo básico.

Android permite la captura de imágenes y vídeo a través de la API `android.hardware.camera2` o con el `Intent` de la cámara. Las clases principales son las siguientes:

`android.hardware.camera2`	API principal para el control de las cámaras del móvil. Para tomar fotos o videos.
`Camera`	API obsoleta para el control de las cámaras del dispositivo.
`SurfaceView`	Se utiliza para presentar una vista previa de cámara en directo para el usuario.
`MediaRecorder`	Para grabar vídeo de la cámara.
`Intent`	Un tipo de intento de `MediaStore.ACTION_IMAGE_CAPTURE` o `MediaStore.ACTION_VIDEO_CAPTURE` que se utiliza para capturar imágenes o vídeos sin utilizar el objeto `Camera`.

Las declaraciones en el Manifest

Antes de iniciar el desarrollo de su aplicación con la API de la cámara, se debe asegurar que el manifiesto tiene las declaraciones adecuadas para el uso de hardware de la cámara y otras funciones relacionadas.
Permisos de la cámara. La aplicación debe solicitar permiso para utilizar una cámara del dispositivo.

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

Nota: Si se utiliza la cámara vía un `Intent`, la aplicación no necesita solicitar este permiso.

Características de la cámara. La aplicación también debe declarar el uso de funciones de la cámara, por ejemplo:

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

La adición de funciones de la cámara al manifiesto ocasiona que Google Play evite que la aplicación se instale en dispositivos que no incluyan una cámara o no sean compatibles con las funciones de la cámara que se especifica.

Si la aplicación utiliza una cámara o una característica de la cámara para un función particular, pero no lo requiere, se debe especificar esto en el manifiesto mediante la inclusión del atributo `android:required`, y asignarlo en `false`:

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

Permiso de almacenamiento. Si la aplicación guarda las imágenes o videos en un almacenamiento externo del dispositivo (tarjeta SD), también debe especificar en el manifiesto.

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

Permiso de grabación de audio. Para grabar audio con captura de video, se debe solicitar el permiso de captura de audio.

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

Permiso de ubicación. Si la aplicación etiqueta imágenes con información de ubicación GPS, se debe solicitar permiso de ubicación:

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

Uso de las aplicaciones de la cámara.

Una forma rápida para la toma de imágenes o vídeos en la aplicación, sin necesidad de una gran cantidad de código adicional, es el uso de un `Intent` para invocar una aplicación de la cámara existente. Un intento a la cámara hace una petición para capturar una imagen o un videoclip a través de una aplicación de cámara existente y luego devuelve el control a la aplicación. El procedimiento para invocar un intento cámara sigue estos pasos generales:

1. Crear un Camera `Intent`. Crea un `Intent` que solicite una imagen o video, utilizando uno de los siguientes intentos:

`MediaStore.ACTION_IMAGE_CAPTURE`. Tipo de `Intent` para solicitar la imagen a la aplicación con la cámara.

`MediaStore.ACTION_VIDEO_CAPTURE`. Tipo de `Intent` para solicitar un video a la aplicación con la cámara.

2. Iniciar el `Camera Intent`. Utilizar el método `startActivityForResult`() para ejecutar el intento de la cámara. Después de iniciar el intento, la aplicación se muestra la interface de usuario en la pantalla y el usuario puede tomar fotos o video.
3. Recepción del `Intent Result`. Configurar el método `onActivityResult`() de la aplicación para recibir la respuesta y los datos del intento de la cámara. Cuando el usuario termina, o cancela la operación, el sistema invoca a este método.

El intento para la captura de la imagen.

La captura de imágenes con un intento de la cámara es la manera rápida para que la aplicación tome fotos con la codificación mínima. Un intento de captura de imágenes puede incluir la siguiente información adicional:

`MediaStore.EXTRA_OUTPUT`. Esta configuración requiere un objeto URI que especifique un nombre de archivo y la ruta en la que desea guardar la imagen. Esta configuración es opcional pero muy recomendada. Si no se especifica este valor, la aplicación de la cámara guarda la imagen solicitada en la ubicación predeterminada con un nombre predeterminado, especificado en el campo `Intent.getData`() del intento.

El siguiente ejemplo demuestra cómo construir un intento de captura de imagen y ejecutarlo.

```
private static final int CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE = 100;
private Uri fileUri;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    // create Intent to take a picture and return control to the calling application
    Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    fileUri = getOutputMediaFileUri(MEDIA_TYPE_IMAGE); // create a file to save the image
    intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri); // set the image file name
    // start the image capture Intent
    startActivityForResult(intent, CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE);
}
```

Cuando se ejecuta el método `startActivityForResult()`, los usuarios ven la interfaz de aplicación de la cámara. Después de que el usuario toma una foto (o se cancela la operación), la interfaz de usuario vuelve a la aplicación, y se debe interceptar el método `onActivityResult()` para recibir el resultado del intento y continuar la ejecución de la aplicación.

El intento para la captura de vídeo.

La captura de vídeo utilizando un intento de la cámara es una forma rápida para grabar vídeos con la codificación mínima. Un intento de captura de vídeo puede incluir la siguiente información adicional:

`MediaStore.EXTRA_OUTPUT`. Esta configuración requiere un URI que especifica un nombre de archivo y la ruta en la que desea guardar el vídeo. Esta configuración es opcional pero muy recomendada. Si no se especifica este valor, la aplicación de la cámara guarda el vídeo solicitado en la ubicación predeterminada con un nombre predeterminado, especificado en el campo `Intent.getData()`.

`MediaStore.EXTRA_VIDEO_QUALITY`. Este valor puede ser 0 para la calidad más baja y tamaño de archivo más pequeño, o 1 para la más alta calidad y el tamaño de archivo más grande.

`MediaStore.EXTRA_DURATION_LIMIT`. Establecer este valor para limitar la duración, en segundos, del vídeo que se está capturado.

`MediaStore.EXTRA_SIZE_LIMIT`. Establecer este valor para limitar el tamaño del archivo, en bytes, del vídeo que se está capturado.

El siguiente ejemplo demuestra cómo construir una captura de la intención de vídeo y ejecutarlo.

```
private static final int CAPTURE_VIDEO_ACTIVITY_REQUEST_CODE = 200;
private Uri fileUri;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    //create new Intent
    Intent intent = new Intent(MediaStore.ACTION_VIDEO_CAPTURE);
    fileUri = getOutputMediaFileUri(MEDIA_TYPE_VIDEO);  // create file to save the video
    intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri);  // set the image file name
    intent.putExtra(MediaStore.EXTRA_VIDEO_QUALITY, 1); // set video image quality to
high
    // start the Video Capture Intent
    startActivityForResult(intent, CAPTURE_VIDEO_ACTIVITY_REQUEST_CODE);
}
```

Cuando se ejecuta el método `startActivityForResult ()`, los usuarios ven la interfaz modificada de aplicación de la cámara. Después de que el usuario termina de grabar un video (o se cancela la operación), la interfaz de usuario vuelve a la aplicación, y debe interceptar el método `onActivityResult ()` para recibir el resultado de la intención y continuar la ejecución de la aplicación.

Recepción el resultado de la intención de la cámara.

Una vez que se ha construido y ejecutado el intento de cámara para una imagen o vídeo, la aplicación debe estar configurada para recibir el resultado de la intención. Con el fin de recibir el resultado de la intención, es necesario sustituir el `onActivityResult ()` en la actividad que se inició el intento. El siguiente ejemplo muestra cómo reemplazar `onActivityResult()` para capturar el resultado de los intentos de la imagen y vídeo.

```
private static final int CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE = 100;
private static final int CAPTURE_VIDEO_ACTIVITY_REQUEST_CODE = 200;
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE) {
        if (resultCode == RESULT_OK) {
            // Image captured and saved to fileUri specified in the Intent
            Toast.makeText(this, "Image saved to:\n" +
```

```
                data.getData(), Toast.LENGTH_LONG).show();
        } else if (resultCode == RESULT_CANCELED) {
            // User cancelled the image capture
        } else {
            // Image capture failed, advise user
        }
    }
    if (requestCode == CAPTURE_VIDEO_ACTIVITY_REQUEST_CODE) {
        if (resultCode == RESULT_OK) {
            // Video captured and saved to fileUri specified in the Intent
            Toast.makeText(this, "Video saved to:\n" +
                data.getData(), Toast.LENGTH_LONG).show();
        } else if (resultCode == RESULT_CANCELED) {
            // User cancelled the video capture
        } else {
            // Video capture failed, advise user
        }
    }
}
```

Una vez que la actividad recibe un resultado exitoso, la imagen o vídeo capturado se encuentra disponible en la ubicación especificada para que la aplicación la acceda.

Construcción de una aplicación con la cámara.
Algunos desarrolladores pueden requerir una interfaz de usuario de la cámara que se adapte a la apariencia de su aplicación o proporcione características especiales. La creación de una actividad personalizada de la cámara requiere más código que utilizando un intento, pero se puede tener una experiencia más atractiva para los usuarios.

Nota: La siguiente guía es para la API obsoleta de la cámara. Para las aplicaciones nuevas o avanzadas de la cámara, se recomienda la API `android.hardware.camera2` más reciente.

Los pasos para crear una interfaz personalizada de la cámara son los siguientes:

- **Detectar y Acceder a la Cámara**. Crear código para comprobar la existencia de las cámaras y la petición de acceso.
- **Crear una clase de vista previa**. Crear una clase de vista previa de la cámara que herede de `SurfaceView` e implante la interfaz `SurfaceHolder`. Esta clase previsualiza las imágenes en directo desde la cámara.
- **Crear una plantilla de vista previa**. Una vez que se tenga la clase de vista previa de lacámara, crear una plantilla que incorpore la vista previa y los controles de interfaz de usuario que se desee.
- **Configurar los escuchas para la captura**. Conectar los escuchas de los controles de la interfaz para iniciar las capturas de imágenes o vídeo en respuesta a las acciones del usuario, como digitar un botón.
- **Capturar y guardar archivos**. Configurar el código para capturar imágenes o vídeos y guardar la salida.
- **Liberar la cámara**. Después de usar la cámara, la aplicación debe liberarla adecuadamente para su uso por otras aplicaciones.

El hardware de la cámara es un recurso compartido que debe ser manejado con cuidado para que la aplicación no choque con otras aplicaciones que también deseen usarla.

Precaución: Recordar que se debe liberar el objeto de la cámara con la llamada a `Camera.release()` cuando la aplicación termina su uso. Si la aplicación no libera correctamente la cámara, todos los intentos posteriores para acceder a la cámara, incluyendo los de la propia aplicación, producirán un error y pueden causar que la aplicación, u otra, se cierren.

Detección del hardware de la cámara.

Si la aplicación no requiere específicamente una cámara usando la declaración en el manifiesto, se debe comprobar si la cámara está disponible en tiempo de ejecución. Para realizar esta comprobación, utilizar el método `PackageManager.hasSystemFeature()`, como se muestra en el código de ejemplo a continuación:

```
/** Check if this device has a camera */
private boolean checkCameraHardware(Context context) {
    if (context.getPackageManager().hasSystemFeature(PackageManager.FEATURE_CAMERA)){
        // this device has a camera
        return true;
    } else {
        // no camera on this device
        return false;
    }
}
```

Los dispositivos Android pueden tener múltiples cámaras, por ejemplo, una cámara de frente para fotografía y una cámara frontal para video. Android 2.3 (API Nivel 9), o superiores, permiten comprobar el número de cámaras disponibles en un dispositivo utilizando el método `Camera.getNumberOfCameras()`.

Acceso a las cámaras.

Si se ha detectado que el dispositivo tiene una cámara, se debe solicitar su acceso para conseguir una instancia de la cámara (a menos que se utilice un intento para acceder a la cámara).

Para acceder a la cámara principal, se utiliza el método Camera.open() y asegurarse de atrapar alguna excepción, como se muestra en el siguiente código:

```
/** A safe way to get an instance of the Camera object. */
public static Camera getCameraInstance(){
    Camera c = null;
    try {
        c = Camera.open(); // attempt to get a Camera instance
    }
    catch (Exception e){
        // Camera is not available (in use or does not exist)
    }
    return c; // returns null if camera is unavailable
}
```

Precaución: Comprobar si hay excepciones cuando se utiliza `Camera.open()`. El no comprobar las excepciones si la cámara está en uso, o no existe, provocará que la solicitud se cierre por el sistema.

En los dispositivos con Android 2.3 (API Nivel 9), o superior, se puede acceder a las cámaras específicas utilizando `Camera.open(int)`. El código del ejemplo anterior accederá a la primera, cámara frontal en el dispositivo con más de una cámara.

Comprobación de las funciones de la cámara.

Una vez que se tenga acceso a una cámara, se puede obtener más información sobre sus capacidades utilizando el método `Camera.getParameters()` y comprobar los objetos `Camera.Parameters` devueltos con las capacidades soportadas. Al usar API Nivel 9, o superior, utilizar `Camera.getCameraInfo ()` para determinar si una cámara está en la parte delantera o trasera del dispositivo, y la orientación de la imagen.

Crear una clase de previsualización

Para que los usuarios tomen fotos o video con eficacia, deben ser capaces de ver lo que ve la cámara del dispositivo. Una clase de vista previa de la cámara es `SurfaceView` que puede mostrar los datos de imágenes en vivo procedentes de una cámara, por lo que los usuarios pueden encuadrar y capturar una imagen o video.

El siguiente código de ejemplo muestra cómo crear una clase básica de la cámara de vista previa que se pueda incluir en una plantilla `View`. Esta clase implanta `SurfaceHolder.Callback` con el fin de capturar los eventos de llamada para crear y destruir la vista, que son necesarios para la asignación de la entrada de cámara de vista previa.

```
/** A basic Camera preview class */
public class CameraPreview extends SurfaceView implements SurfaceHolder.Callback {
    private SurfaceHolder mHolder;
    private Camera mCamera;
    public CameraPreview(Context context, Camera camera) {
        super(context);
        mCamera = camera;
        // Install a SurfaceHolder.Callback so we get notified when the
        // underlying surface is created and destroyed.
        mHolder = getHolder();
        mHolder.addCallback(this);
        // deprecated setting, but required on Android versions prior to 3.0
        mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    }
    public void surfaceCreated(SurfaceHolder holder) {
        // The Surface has been created, now tell the camera where to draw the preview.
        try {
            mCamera.setPreviewDisplay(holder);
            mCamera.startPreview();
        } catch (IOException e) {
            Log.d(TAG, "Error setting camera preview: " + e.getMessage());
        }
    }
    public void surfaceDestroyed(SurfaceHolder holder) {
        // empty. Take care of releasing the Camera preview in your activity.
    }
    public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {
        // If your preview can change or rotate, take care of those events here.
        // Make sure to stop the preview before resizing or reformatting it.
        if (mHolder.getSurface() == null){
          // preview surface does not exist
          return;
        }
        // stop preview before making changes
        try {
            mCamera.stopPreview();
        } catch (Exception e){
          // ignore: tried to stop a non-existent preview
        }
        // set preview size and make any resize, rotate or
        // reformatting changes here
        // start preview with new settings
        try {
            mCamera.setPreviewDisplay(mHolder);
            mCamera.startPreview();
        } catch (Exception e){
            Log.d(TAG, "Error starting camera preview: " + e.getMessage());
        }
    }
}
```

Si desea establecer un tamaño específico para la previa de la cámara, ajustarlo en el método `surfaceChanged`() como se indica en los comentarios anteriores. Al establecer el tamaño de la vista previa, se debe utilizar los valores de `getSupportedPreviewSizes`(). No establecer valores arbitrarios en el método `setPreviewSize`().

Colocación de previsualización en una plantilla.
Una clase de vista previa de la cámara, como el ejemplo mostrado en la sección anterior, se debe colocar en la plantilla de una actividad junto con otros controles de interfaz de usuario para tomar una foto o vídeo.

El siguiente código de la plantilla proporciona una vista muy básica que se puede utilizar para mostrar una vista previa de la cámara. En este ejemplo, el elemento `FrameLayout` es el recipiente para la clase de la cámara de vista previa. Este tipo de diseño se utiliza para que la información o los controles de imagen adicionales pueden ser superpuestas sobre las imágenes de la cámara de vista previa en vivo.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
  <FrameLayout
    android:id="@+id/camera_preview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="1" />
  <Button
    android:id="@+id/button_capture"
    android:text="Capture"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center" />
</LinearLayout>
```

En la mayoría de los dispositivos, la orientación predeterminada de la previa de la cámara es la horizontal. Esta plantilla de ejemplo especifica un diseño horizontal (paisaje) y el código de abajo corrige la orientación de la aplicación de paisaje. Por simplicidad en el renderizado de una vista previa de la cámara, se debe cambiar la orientación de la actividad de vista previa de la aplicación a horizontal añadiendo lo siguiente al manifiesto.

```
<activity android:name=".CameraActivity"
          android:label="@string/app_name"
          android:screenOrientation="landscape">
          <!-- configure this activity to use landscape orientation -->
          <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Nota: Una vista previa de la cámara no tiene que estar en modo horizontal. A partir de Android 2.2 (API Nivel 8), se puede utilizar el método setDisplayOrientation() para establecer la rotación de la imagen de vista previa. Con el fin de cambiar la orientación de vista previa cuando el usuario reorienta el teléfono, dentro del método surfaceChanged() de la clase de vista previa, la primera parada de la vista previa con Camera.stopPreview(), cambiar la orientación y luego iniciar la vista previa de nuevo con Camera.startPreview().

En la actividad de la vista de la cámara, añadir la clase de vista previa al elemento FrameLayout que se muestra en el ejemplo anterior. La actividad de la cámara también debe garantizar que se libera cuando la cámara está en pausa o se apague. El siguiente ejemplo muestra cómo modificar una actividad de la cámara para pegar la clase de vista previa.

```
public class CameraActivity extends Activity {
```

```
    private Camera mCamera;
    private CameraPreview mPreview;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // Create an instance of Camera
        mCamera = getCameraInstance();
        // Create our Preview view and set it as the content of our activity.
        mPreview = new CameraPreview(this, mCamera);
        FrameLayout preview = (FrameLayout) findViewById(R.id.camera_preview);
        preview.addView(mPreview);
    }
}
```

Nota: El método `getCameraInstance`() en el ejemplo anterior se refiere al método de ejemplo que se muestra en Acceso a las cámaras .

Captura de imágenes.
Una vez que haya construido una clase de vista previa y una plantilla para mostrarla, se está listo para iniciar la captura de imágenes con la aplicación. En el código de aplicación, se debe configurar los escuchas para los controles de la interfaz de usuario para responder a una acción del usuario por tomar una foto.

Con el fin de recuperar una imagen, utilizar el método `Camera.takePicture`(). Este método toma tres parámetros que reciben los datos de la cámara. Con el fin de recibir datos en un formato JPEG, se debe implantar una interfaz `Camera.PictureCallback` para recibir los datos de imagen y escribir en un archivo. El código siguiente muestra una implantación básica de la interfaz `Camera.PictureCallback` para guardar una imagen recibida de la cámara.

```
private PictureCallback mPicture = new PictureCallback() {
    @Override
    public void onPictureTaken(byte[] data, Camera camera) {
        File pictureFile = getOutputMediaFile(MEDIA_TYPE_IMAGE);
        if (pictureFile == null){
            Log.d(TAG, "Error creating media file, check storage permissions: " +
                e.getMessage());
            return;
        }
        try {
            FileOutputStream fos = new FileOutputStream(pictureFile);
            fos.write(data);
            fos.close();
        } catch (FileNotFoundException e) {
            Log.d(TAG, "File not found: " + e.getMessage());
        } catch (IOException e) {
            Log.d(TAG, "Error accessing file: " + e.getMessage());
        }
    }
};
```

Tomar la captura de una imagen mediante una llamada al método Camera.takePicture(). El siguiente código de ejemplo muestra cómo llamar a este método desde un botón `View.OnClickListener`.

Nota: El miembro `mPicture` en el ejemplo siguiente se refiere al código ejemplo anterior.
Precaución: Recordar que se debe liberar el objeto de la cámara mediante una llamada a `Camera.release`() cuando la aplicación termina de usarlo!

Captura de vídeos.
La captura de vídeo utilizando la referencia de Android requiere un manejo cuidadoso del objeto de la cámara y la coordinación con la clase `MediaRecorder`. Durante la grabación de vídeo con la cámara, se debe administrar el `Camera.lock`() y `Camera.unlock`() para permitir que `MediaRecorder` acceda al hardware de la cámara, además a `Camera.open`() y `Camera.release`().

Nota: A partir de Android 4.0 (API de nivel 14), las llamadas a `Camera.lock`() y `Camera.unlock`()se gestionan de forma automática.

A diferencia de la toma de fotografías con la cámara del dispositivo, la captura de vídeo requiere una llamada muy particular. Se debe seguir un orden específico de ejecución para preparar con éxito la captura de vídeo, como se detalla a continuación.

1. **Abrir la cámara**. Utilizar `Camera.open`() para obtener una instancia del objeto de cámara.
2. **Conectar la vista previa**. Preparar una vista previa de la imagen de la cámara en vivo conectando un `SurfaceView` a la cámara mediante `Camera.setPreviewDisplay`().
3. **Iniciar vista previa.** Invocar a `Camera.startPreview`() para comenzar la presentación de las imágenes de la cámara en vivo.
4. **Iniciar la grabación de vídeo**. Los siguientes pasos deben ser en orden para grabar vídeo con éxito:
   a. **Desbloquear la cámara**. Desbloquear cámara para uso de `MediaRecorder` invocando a `Camera.unlock`().
   b. **Configurar el MediaRecorder**. Llamar a los siguientes métodos `MediaRecorder` en este orden.
      1. `setCamera`(). Configurar la cámara para utilizar la captura de vídeo, utilizar la instancia actual de `Camera`.
      2. `setAudioSource`(). Establecer la fuente de audio, utilizar `MediaRecorder.AudioSource.CAMCORDER`.
      3. `setVideoSource`(). Establecer la fuente de vídeo, utilizar `MediaRecorder.VideoSource.CAMERA`.
      4. Ajustar el formato de salida de vídeo y codificación. Para Android 2.2 (API Nivel 8) y superior, utilizar el método `MediaRecorder.setProfile`, y obtener un perfil de instancia utilizando `CamcorderProfile.get`(). Para las versiones de Android anteriores a 2.2, configurar los parámetros de formato y codificación de salida de vídeo:
         i. `setOutputFormat`(). Ajustar el formato de salida, especificar la configuración predeterminada o `MediaRecorder.OutputFormat.MPEG_4`.
         ii. `setAudioEncoder`(). Establecer el tipo de codificación de sonido, especificar la configuración predeterminada o `MediaRecorder.AudioEncoder.AMR_NB`.
         iii. `setVideoEncoder`(). Establecer el tipo de codificación de vídeo, especificar la configuración predeterminada o `MediaRecorder.VideoEncoder.MPEG_4_SP`.
      5. `setOutputFile`(). Establecer el archivo de salida, utilizar `getOutputMediaFile(MEDIA_TYPE_VIDEO).toString()` del método de ejemplo de la sección guardar los archivos multimedia.
      6. `setPreviewDisplay`(). Especificar el elemento de la plantilla de vista previa `SurfaceView` para la aplicación. Utilizar el mismo objeto que especificado para `Connect Preview`.
         Precaución: Se debe llamar a estos métodos de configuración `MediaRecorder` en este orden, de lo contrario la aplicación encontrará errores y la grabación fallará.
   c. **Preparar MediaRecorder**. Preparar el `MediaRecorder` con valores de configuración proporcionados por llamar `MediaRecorder.prepare` ().
   d. **Comience MediaRecorder**. Iniciar la grabación de vídeo llamando `MediaRecorder.start` ().
5. **Detener la grabación de vídeo**. Llame a los métodos siguientes en orden, para completar con éxito una grabación de vídeo:

a. **Detener `MediaRecorder`**. Detener la grabación de vídeo llamando a `MediaRecorder.stop()`.
b. **Restablecer `MediaRecorder`**. De forma opcional, retire los ajustes de configuración de la grabadora llamando `MediaRecorder.reset()`.
c. **Liberar `MediaRecorder`**. Soltar el `MediaRecorder` llamando `MediaRecorder.release ()`.
d. **Bloquear la cámara**. Bloquear la cámara para que las futuras sesiones de `MediaRecorder` pueden utilizarla llamando `Camera.lock()`. A partir de Android 4.0 (API de nivel 14), no se requiere esta llamada a menos que el `MediaRecorder.prepare()` falle.

6. **Detener la vista previa**. Cuando su actividad ha terminado de usar la cámara, detener la vista previa utilizando `Camera.stopPreview()`.
7. **Liberar la cámara**. Liberar la cámara para que otras aplicaciones puedan utilizarla llamando `Camera.release()`.

Nota: Es posible utilizar MediaRecorder sin crear primero una vista previa de la cámara y saltar los primeros pasos de este proceso.
Sugerencia: Si la aplicación se utiliza típicamente para la grabación de vídeo, ajustar `setRecordingHint(booleano)` a `true` antes de comenzar la previsualización. Esta configuración ayuda a reducir el tiempo que se necesita para iniciar la grabación.

Configuración de MediaRecorder.
El siguiente código de ejemplo muestra cómo configurar y preparar la clase `MediaRecorder` para la grabación de vídeo correctamente:

```
private boolean prepareVideoRecorder(){
    mCamera = getCameraInstance();
    mMediaRecorder = new MediaRecorder();
      // Step 1: Unlock and set camera to MediaRecorder
    mCamera.unlock();
    mMediaRecorder.setCamera(mCamera);
      // Step 2: Set sources
    mMediaRecorder.setAudioSource(MediaRecorder.AudioSource.CAMCORDER);
    mMediaRecorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);
      // Step 3: Set a CamcorderProfile (requires API Level 8 or higher)
    mMediaRecorder.setProfile(CamcorderProfile.get(CamcorderProfile.QUALITY_HIGH));
      // Step 4: Set output file
    mMediaRecorder.setOutputFile(getOutputMediaFile(MEDIA_TYPE_VIDEO).toString());
      // Step 5: Set the preview output
    mMediaRecorder.setPreviewDisplay(mPreview.getHolder().getSurface());
      // Step 6: Prepare configured MediaRecorder
    try {
        mMediaRecorder.prepare();
    } catch (IllegalStateException e) {
        Log.d(TAG, "IllegalStateException preparing MediaRecorder: " + e.getMessage());
        releaseMediaRecorder();
        return false;
    } catch (IOException e) {
        Log.d(TAG, "IOException preparing MediaRecorder: " + e.getMessage());
        releaseMediaRecorder();
        return false;
    }
    return true;
}
```

Antes de Android 2.2 (API Nivel 8), se debe configurar los parámetros de formatos de salida y codificación directa, en lugar de utilizar `CamcorderProfile`. Este enfoque se muestra en el código siguiente:

```
// Step 3: Set output format and encoding (for versions prior to API Level 8)
```

```
mMediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
mMediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.DEFAULT);
mMediaRecorder.setVideoEncoder(MediaRecorder.VideoEncoder.DEFAULT);
```

Los siguientes parámetros de grabación de vídeo de `MediaRecorder` dan ajustes predeterminados, sin embargo, es posible ajustar estas configuraciones para la aplicación:

- `setVideoEncodingBitRate()`
- `setVideoSize()`
- `setVideoFrameRate()`
- `setAudioEncodingBitRate()`
- `setAudioChannels()`
- `setAudioSamplingRate()`

Inicio y paro de `MediaRecorder`.

Al iniciar y detener la grabación de vídeo utilizando la clase `MediaRecorder`, se debe seguir un orden específico, que se enumera a continuación.

1. Desbloquear la cámara con `Camera.unlock`().
2. Configurar `MediaRecorder` como se muestra en el ejemplo anterior.
3. Iniciar la grabación usando `MediaRecorder.start`().
4. Grabar el vídeo.
5. Detener la grabación usando `MediaRecorder.stop`().
6. Liberar la grabadora multimedia con `MediaRecorder.release`().
7. Bloquear la cámara usando `Camera.lock`().

El siguiente código de ejemplo muestra cómo conectar un botón para iniciar y detener la grabación de vídeo utilizando la cámara y la clase `MediaRecorder` correctamente.

Nota: Al completar una grabación de vídeo, no liberar la cámara o de lo contrario se detendrá la vista previa.

```
private boolean isRecording = false;
// Add a listener to the Capture button
Button captureButton = (Button) findViewById(id.button_capture);
captureButton.setOnClickListener(
    new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if (isRecording) {
                // stop recording and release camera
                mMediaRecorder.stop();  // stop the recording
                releaseMediaRecorder(); // release the MediaRecorder object
                mCamera.lock();         // take camera access back from MediaRecorder
                // inform the user that recording has stopped
                setCaptureButtonText("Capture");
                isRecording = false;
            } else {
                // initialize video camera
                if (prepareVideoRecorder()) {
                    // Camera is available and unlocked, MediaRecorder is prepared,
                    // now you can start recording
                    mMediaRecorder.start();
                    // inform the user that recording has started
                    setCaptureButtonText("Stop");
                    isRecording = true;
                } else {
```

```
                    // prepare didn't work, release the camera
                    releaseMediaRecorder();
                    // inform user
                }
            }
        }
    }
);
```

Liberación de la cámara.

Para liberar una instancia del objeto `Camera`, utilizar el método `Camera.release()`, como se muestra en el código de ejemplo a continuación.

```
public class CameraActivity extends Activity {
    private Camera mCamera;
    private SurfaceView mPreview;
    private MediaRecorder mMediaRecorder;
      :
    @Override
    protected void onPause() {
        super.onPause();
        releaseMediaRecorder();       // if you are using MediaRecorder, release it first
        releaseCamera();              // release the camera immediately on pause event
    }
    private void releaseMediaRecorder(){
        if (mMediaRecorder != null) {
            mMediaRecorder.reset();   // clear recorder configuration
            mMediaRecorder.release(); // release the recorder object
            mMediaRecorder = null;
            mCamera.lock();           // lock camera for later use
        }
    }
    private void releaseCamera(){
        if (mCamera != null){
            mCamera.release();        // release the camera for other applications
            mCamera = null;
        }
    }
}
```

Guardar los archivos de multimedios.

El siguiente código de ejemplo muestra cómo crear una ubicación de `File` o `URI` de un archivo multimedia que se puede utilizar cuando se invoca la cámara de un dispositivo con un `Intent` o como parte de una aplicación de la cámara.

```
public static final int MEDIA_TYPE_IMAGE = 1;
public static final int MEDIA_TYPE_VIDEO = 2;
/** Create a file Uri for saving an image or video */
private static Uri getOutputMediaFileUri(int type){
      return Uri.fromFile(getOutputMediaFile(type));
}
/** Create a File for saving an image or video */
private static File getOutputMediaFile(int type){
    // To be safe, you should check that the SDCard is mounted
    // using Environment.getExternalStorageState() before doing this.
    File mediaStorageDir = new File(Environment.getExternalStoragePublicDirectory(
              Environment.DIRECTORY_PICTURES), "MyCameraApp");
    // This location works best if you want the created images to be shared
    // between applications and persist after your app has been uninstalled.
```

```
    // Create the storage directory if it does not exist
    if (! mediaStorageDir.exists()){
        if (! mediaStorageDir.mkdirs()){
            Log.d("MyCameraApp", "failed to create directory");
            return null;
        }
    }
    // Create a media file name
    String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
    File mediaFile;
    if (type == MEDIA_TYPE_IMAGE){
        mediaFile = new File(mediaStorageDir.getPath() + File.separator +
        "IMG_"+ timeStamp + ".jpg");
    } else if(type == MEDIA_TYPE_VIDEO) {
        mediaFile = new File(mediaStorageDir.getPath() + File.separator +
        "VID_"+ timeStamp + ".mp4");
    } else {
        return null;
    }
    return mediaFile;
}
```

Características de la cámara.

Android es compatible con una amplia gama de funciones de la cámara que se pueden controlar con la aplicación de cámara, como el formato de imagen, modo de flash, ajustes de enfoque, y muchos más. La mayoría de las funciones de la cámara se acceden y establecen con el objeto `Camera.Parameters`. Sin embargo, hay varias características importantes que requieren más que simples ajustes en `Camera.Parameters`.

Tabla 1. Características comunes de la cámara según el nivel de la API de Android en el que se introdujeron.

Característica de API	Level	Descripción
`Face Detection`	14	Identifica caras humanas en la imagen y las usa para el foco, medición y balance de blancos.
`Metering Areas`	14	Especifica una o más áreas en una imagen para cálculo de balance de blancos.
`Focus Areas`	14	Asigna una o más áreas de una imagen para el uso del foco.
`White Balance Lock`	14	Detiene o inicia el ajuste automático del balance de blancos.
`Exposure Lock`	14	Detiene o inicia los ajustes automáticos de exposición.
`Video Snapshot`	14	Toma una foto mientras toma video (`frame grab`).
`Time Lapse Video`	11	Graba tramas con retardos para grabar un video con lapso de tiempo.
`Multiple Cameras`	9	Permite más de una cámara, incluyendo la cámara frontal y trasera.
`Focus Distance`	9	Reporta distancias entre la cámara y los objetos que parecen estar enfocados.
`Zoom`	8	Configura la ampliación de la imagen.
`Exposure Compensation`	8	Incrementa o decrementa el nivel de exposición de luz.
`GPS Data`	5	Incluye u omite los datos de la posición geográfica con la imagen.
`White Balance`	5	Configura el balance blanco, que afecta los valores del color en la imagen capturada.
`Focus Mode`	5	Establecer cómo la cámara se centra en un tema como automática, fija, macro o infinito.
`Scene Mode`	5	Aplicar un modo predeterminado para tipos específicos de la fotografía nocturna, playa, nieve o escenas de luz de velas.
`JPEG Quality`	5	Ajusta el nivel de compresión de una imagen JPEG, lo que aumenta o disminuye la calidad y tamaño del archivo de la imagen.
`Flash Mode`	5	Apaga el flash, lo apaga o utiliza la configuración automática.
`Color Effects`	5	Aplica efectos de color a la imagen capturada, blanco y negro, tono sepia o negativo.
`Anti-Banding`	5	Reduce el efecto de bandas en gradientes de color debido a la compresión JPEG.

`Picture Format`	1	Especificar el formato de archivo de la imagen.
`Picture Size`	1	Especificar las dimensiones en píxeles de la imagen guardada.

Nota: Estas características no son compatibles con todos los dispositivos debido a las diferencias de hardware y software de aplicación.

Verificación de la disponibilidad de las funciones.
Se puede comprobar la disponibilidad de las características de la cámara con una instancia del objeto con los parámetros de la cámara, y la comprobación de los métodos pertinentes. El siguiente ejemplo de código se muestra cómo obtener un objeto `Camera.Parameters` y comprueba si la cámara es compatible con la función de enfoque automático:

```
// get Camera parameters
Camera.Parameters params = mCamera.getParameters();
List<String> focusModes = params.getSupportedFocusModes();
if (focusModes.contains(Camera.Parameters.FOCUS_MODE_AUTO)) {
  // Autofocus mode is supported
}
```

Uso de las características de la cámara.
La mayoría de las funciones de la cámara se activan y controlan mediante un objeto `Camera.Parameters` . Se obtiene este objeto con una instancia del objeto de la cámara, llamando al método `getParameters`(), cambiando el objeto de parámetro devuelto, y luego ponerlo de nuevo en el objeto de la cámara, como se demuestra en el siguiente código de ejemplo:

```
// get Camera parameters
Camera.Parameters params = mCamera.getParameters();
// set the focus mode
params.setFocusMode(Camera.Parameters.FOCUS_MODE_AUTO);
// set Camera parameters
mCamera.setParameters(params);
```

Áreas de medición y de enfoque.
Las áreas de medición y enfoque trabajan similarmente a otras funciones de la cámara, ya que se controlan a través de métodos en el objeto `Camera.Parameters`. El código siguiente muestra cómo establecer dos zonas de medición de luz para una instancia de la cámara:

```
// Create an instance of Camera
mCamera = getCameraInstance();
// set Camera parameters
Camera.Parameters params = mCamera.getParameters();
if (params.getMaxNumMeteringAreas() > 0){ // check that metering areas are supported
    List<Camera.Area> meteringAreas = new ArrayList<Camera.Area>();
    Rect areaRect1 = new Rect(-100, -100, 100, 100);    // specify an area in center of
image
    meteringAreas.add(new Camera.Area(areaRect1, 600)); // set weight to 60%
    Rect areaRect2 = new Rect(800, -1000, 1000, -800);  // specify an area in upper right
of image
    meteringAreas.add(new Camera.Area(areaRect2, 400)); // set weight to 40%
    params.setMeteringAreas(meteringAreas);
}
mCamera.setParameters(params);
```

El objeto `Camera.Area` contiene dos parámetros de datos: Un objeto `Rect` para especificar un área dentro del campo de visión de la cámara y un valor de peso, lo que le dice a la cámara qué nivel de importancia de esta zona se debe dar en la medición de luz o enfoque cálculos.

El campo `Rect` en un objeto `Camera.Area` describe una forma rectangular asignada en una unidad de red de `2000x2000`. Las coordenadas `-1000,-1000` representan la esquina superior izquierda de la imagen de la cámara, y las coordenadas `1000,1000` representan la esquina inferior derecha de la imagen de la cámara, como se muestra en la siguiente ilustración.

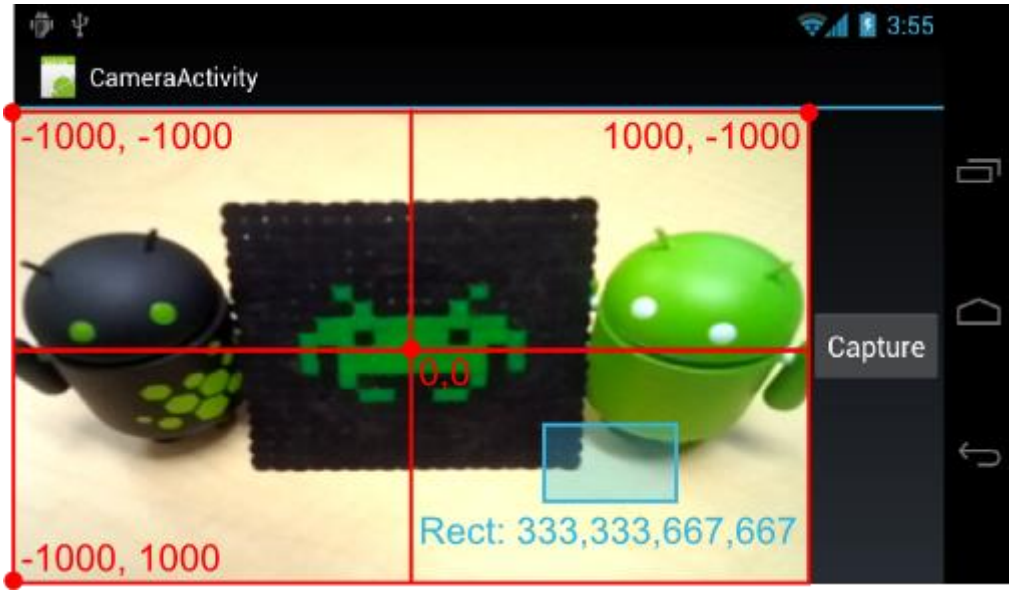


Figura 1. Las líneas rojas muestran el sistema de coordenadas que especifican un `Camera.Area` en una vista previa de la cámara. El cuadro azul muestra la ubicación y la forma del área de la cámara con los valores `333333667667` del `Rect`.

Detección de rostros.

El uso de la función de detección de rostros en la cámara requiere algunos pasos generales:
Verificar que la detección de rostros es compatible con el dispositivo.
Crear un escucha de detección de rostros.
Añadir el escucha de detección de rostros al objeto `Camera`.
Iniciar la detección de rostros después de la vista previa (y después de cada reinicio de vista previa).

La función de detección facial no es compatible con todos los dispositivos. Comprobar que esta función es compatible llamando `getMaxNumDetectedFaces()`. Un ejemplo de este control se muestra en el método de muestreo `startFaceDetection` () a continuación.

Con el fin de ser notificada y responde a la detección de un rostro, la aplicación de la cámara debe establecer un escucha de eventos de detección de rostros. Se debe crear una clase de escucha que implante la interfaz `Camera.FaceDetectionListener` como se muestra en el código de ejemplo a continuación.

```
class MyFaceDetectionListener implements Camera.FaceDetectionListener {
    @Override
    public void onFaceDetection(Face[] faces, Camera camera) {
        if (faces.length > 0){
            Log.d("FaceDetection", "face detected: "+ faces.length +
                    " Face 1 Location X: " + faces[0].rect.centerX() +
                    "Y: " + faces[0].rect.centerY() );
        }
    }
}
```

Después de crear esta clase, se asigna en el objeto la cámara de la aplicación, como se muestra en el código siguiente:

```
mCamera.setFaceDetectionListener(new MyFaceDetectionListener());
```

La aplicación debe iniciar la función de detección de rostros cada vez que se inicia (o reiniciar) la vista previa de la cámara. Crear un método para iniciar la detección de rostros para que se pueda llamar cuando sea necesario, como se muestra a continuación.

```
public void startFaceDetection(){
    // Try starting Face Detection
    Camera.Parameters params = mCamera.getParameters();

    // start face detection only *after* preview has started
    if (params.getMaxNumDetectedFaces() > 0){
        // camera supports face detection, so can start it:
        mCamera.startFaceDetection();
    }
}
```

Se debe iniciar la detección de rostros cada vez que se inicia (o reiniciar) la vista previa de la cámara. Si se utiliza la clase de vista previa, se añade el método `startFaceDetection`(), a los métodos `surfaceCreated` () y `surfaceChanged` ( ) en la clase de vista previa, como se muestra a continuación.

```
public void surfaceCreated(SurfaceHolder holder) {
    try {
        mCamera.setPreviewDisplay(holder);
        mCamera.startPreview();

        startFaceDetection(); // start face detection feature

    } catch (IOException e) {
        Log.d(TAG, "Error setting camera preview: " + e.getMessage());
    }
}
public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {
    if (mHolder.getSurface() == null){
        // preview surface does not exist
        Log.d(TAG, "mHolder.getSurface() == null");
        return;
    }
    try {
        mCamera.stopPreview();
    } catch (Exception e){
        // ignore: tried to stop a non-existent preview
        Log.d(TAG, "Error stopping camera preview: " + e.getMessage());
    }
    try {
        mCamera.setPreviewDisplay(mHolder);
        mCamera.startPreview();
        startFaceDetection(); // re-start face detection feature
    } catch (Exception e){
        // ignore: tried to stop a non-existent preview
        Log.d(TAG, "Error starting camera preview: " + e.getMessage());
    }
}
```

Nota: Recordar llamar a este método después de invocar a `startPreview`(). No intentar iniciar la detección de rostros en el método `onCreate`() de la actividad principal de la aplicación de la cámara, ya que la vista previa no está disponible en este punto en la ejecución de la aplicación.

Video con lapso de tiempo.

El lapso de tiempo de vídeo permite a los usuarios crear clips de vídeo que combinan imágenes tomadas con unos pocos segundos o minutos de diferencia. Esta función utiliza `MediaRecorder` para grabar las imágenes de una secuencia de lapso de tiempo.

Para grabar un video con lapso de tiempo con `MediaRecorder`, se debe configurar el objeto `recorder` como si se estuviera grabando un vídeo normal, establecer los fotogramas capturados por segundo en un número bajo y usar de uno de los ajustes de calidad de lapso de tiempo, como se muestra en el ejemplo de código abajo.

```
// Step 3: Set a CamcorderProfile (requires API Level 8 or higher)
mMediaRecorder.setProfile(CamcorderProfile.get(CamcorderProfile.QUALITY_TIME_LAPSE_HIGH))
;
   :
// Step 5.5: Set the video capture rate to a low number
mMediaRecorder.setCaptureRate(0.1); // capture a frame every 10 seconds
```

Estos ajustes deben realizarse como parte de un procedimiento de configuración mayor de MediaRecorder. Por ejemplo, una vez completada la configuración, se inicia la grabación de vídeo como si estuviera grabando un videoclip normal.

DESARROLLO

EJEMPLO 1.

Paso 1. Crear un nuevo proyecto Camara en Android Studio. En la carpeta java/com.example.mipaquete, abrir y modificar el archivo predeterminado MainActivity.java con el siguiente código:

```
import java.io.*;
import java.util.Date;
import java.text.SimpleDateFormat;
import android.os.*;
import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.provider.MediaStore;
import android.util.Log;
import android.view.View;
import android.widget.Button;
public class MainActivity extends Activity {
    private final String ruta =
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES) +
"/misfotos/";
    private File    f = new File(ruta);
    private Button  jbn;
    @Override
    protected void onCreate(Bundle b) {
        super.onCreate(b);
        setContentView(R.layout.activity_main);
        jbn = (Button) findViewById(R.id.xbn);
        f.mkdirs();
        jbn.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v) {
                String s = ruta + getCode() + ".jpg";
                File f1 = new File(s);
                try{
                    f1.createNewFile();
                }catch(IOException ex){
                    Log.e("Error", "Error:" + ex);
                }
```

```
                Uri u = Uri.fromFile(f1);
                Intent in = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
                in.putExtra(MediaStore.EXTRA_OUTPUT, u);
                startActivityForResult(in, 0);
            }
        });
    }
    private String getCode(){
        SimpleDateFormat sdf = new SimpleDateFormat("yyyymmddhhmmss");
        String n = "pic_" + sdf.format(new Date());;
        return n;
    }
}
```

Paso 2. En la carpeta res/layout, abrir y modificar el archivo activity_main.xml con el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:showIn="@layout/activity_main"
    tools:context=".CamaraActivity">
    <Button
        android:id="@+id/xbn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/foto"/>
</RelativeLayout>
```

Paso 3. En la carpeta res/values, abrir y modificar el archivo strings.xml con el siguiente código:

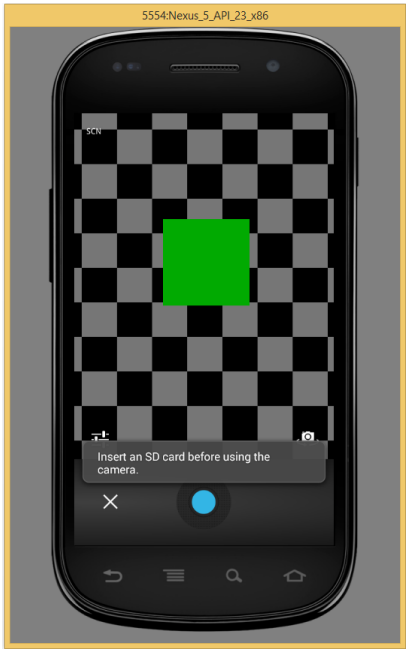
```
<?xml versión="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Camara</string>
    <string name="action_settings">Settings</string>
    <string name="foto">Tomar foto</string>
</resources>
```

Paso 4. En la carpeta app/manifests, abrir y modificar el archivo AndroidManifest.xml para agregar las etiquetas de permisos de uso de la cámara y almacenamiento, entre las etiquetas `<manifest>` y `<application>`, como se muestra enseguida con **letras negritas**:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.escom.camara">
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <application
        :    >
            <activity
                :    >
        :
```

```
            </activity>
    </application>
</manifest>
```

Paso 5. Por último, ejecutar la aplicación. La plantilla solamente contiene un botón para tomar la foto. Al digitar el botón se muestra una animación predeterminada de Android Studio, para indicar que se captura una foto. Si se digita el botón inferior X, para cerrar la imagen de la foto, se regresa a la plantilla principal para tomar otra foto.



- Repetir el ejercicio en un dispositivo real. Verificar en la carpeta Galería, o una equivalente, la foto real tomada.

EJEMPLO 2.

Paso 1. Crear un nuevo proyecto Camara2 en Android Studio. En la carpeta java/com.example.mipaquete, abrir y modificar el archivo predeterminado MainActivity.java con el siguiente código:

```
import java.io.*;
import android.app.Activity;
import android.content.Intent;
import android.graphics.*;
import android.media.MediaScannerConnection;
import android.media.MediaScannerConnection.MediaScannerConnectionClient;
import android.net.Uri;
import android.os.*;
import android.provider.MediaStore;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.*;
public class MainActivity extends Activity {
    Button      jbn;
    RadioButton jrb1, jrb2;
    Intent      i;
    int         TAKE_PICTURE = 1;
    int         SELECT_PICTURE = 2;
    String      s = "";
```

```
    @Override
    public void onCreate(Bundle b){
        super.onCreate(b);
        setContentView(R.layout.activity_main);
        s = Environment.getExternalStorageDirectory() + "/test.jpg";
        jbn = (Button)findViewById(R.id.xbn1);
        jbn.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                jrb1 = (RadioButton)findViewById(R.id.xrb1);
                jrb2 = (RadioButton)findViewById(R.id.xrb2);
                i =  new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
                int code = TAKE_PICTURE;
                if (jrb1.isChecked()) {
                    Uri output = Uri.fromFile(new File(s));
                    i.putExtra(MediaStore.EXTRA_OUTPUT, output);
                } else if (jrb2.isChecked()){
                    i = new Intent(Intent.ACTION_PICK,
android.provider.MediaStore.Images.Media.INTERNAL_CONTENT_URI);
                    code = SELECT_PICTURE;
                }
                startActivityForResult(i, code);
            }
        });
    }
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (requestCode == TAKE_PICTURE) {
            if (data != null) {
                if (data.hasExtra("data")) {
                    ImageView iv = (ImageView)findViewById(R.id.xiv1);
                    iv.setImageBitmap((Bitmap) data.getParcelableExtra("data"));
                }
            } else {
                ImageView iv = (ImageView)findViewById(R.id.xiv1);
                iv.setImageBitmap(BitmapFactory.decodeFile(s));
                new MediaScannerConnectionClient() {
                    private MediaScannerConnection msc = null; {
                        msc = new MediaScannerConnection(getApplicationContext(), this);
msc.connect();
                    }
                    public void onMediaScannerConnected() {
                        msc.scanFile(s, null);
                    }
                    public void onScanCompleted(String path, Uri uri) {
                        msc.disconnect();
                    }
                };
            }
        } else if (requestCode == SELECT_PICTURE){
            Uri image = data.getData();
            InputStream is;
            try {
                is = getContentResolver().openInputStream(image);
                BufferedInputStream bis = new BufferedInputStream(is);
                Bitmap bitmap = BitmapFactory.decodeStream(bis);
                ImageView iv = (ImageView)findViewById(R.id.xiv1);
```

```
                iv.setImageBitmap(bitmap);
            } catch (FileNotFoundException e) {}
        }
    }
}
```

Paso 2. En la carpeta res/layout, abrir y modificar el archivo activity_main.xml con el siguiente código

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    android:id="@+id/widget28"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <Button
        android:text="Toma foto"
        android:id="@+id/xbn1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true">
    </Button>
    <RadioGroup
        android:id="@+id/xrg1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true">
        <RadioButton
            android:id="@+id/xrb0"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Tomar una vista previa">
        </RadioButton>
        <RadioButton
            android:id="@+id/xrb1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Tomar una imagen completa">
        </RadioButton>
        <RadioButton
            android:id="@+id/xrb2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Mostrar una foto de la galería">
        </RadioButton>
    </RadioGroup>
    <ImageView
        android:layout_width="wrap_content"
        android:layout_below="@+id/xrg1"
        android:layout_height="wrap_content"
        android:id="@+id/xiv1">
    </ImageView>
</RelativeLayout>
```

Paso 3. En la carpeta res/values, abrir y modificar el archivo strings.xml con el siguiente código:

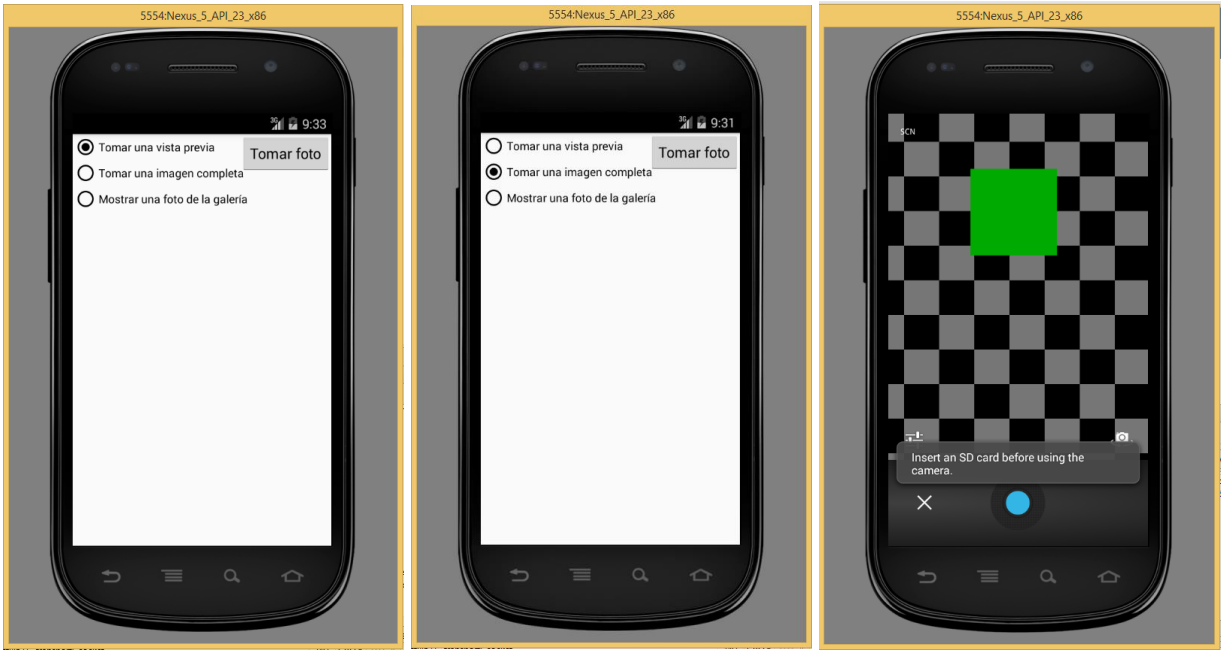
```
<?xml versión=”1.0” encoding=”utf-8”?>
<resources>
    <string name="app_name">Camara</string>
    <string name="action_settings">Settings</string>
    <string name="foto">Tomar foto</string>
</resources>
```

Paso 4. En la carpeta app/manifests, abrir y modificar el archivo AndroidManifest.xml para agregar las etiquetas de permisos de uso de la cámara y almacenamiento, entre las etiquetas <manifest> y <application>, como se muestra enseguida con **letras negritas**:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.escom.camara">
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <application
        :    >
            <activity
                :    >
        :
            </activity>
    </application>
</manifest>
```

Paso 5. Por último, ejecutar la aplicación. La plantilla contiene tres opciones y un botón para tomar la foto.

a. Si se utiliza el emulador virtual, al digitar la selección Tomar una vista previa o Tomar una imagen completa, se muestra una animación predeterminada de Android Studio, para indicar que se captura una foto, en forma similar al ejemplo anterior. Si se digita el botón inferior X, para cerrar la imagen de la foto, se regresa a la plantilla principal para tomar otra foto.



b. Si se digita la selección Mostrar una foto de la galería, se muestra una opción de una carpeta con imágenes:

c. Utilizar el dispositivo real para ejecutar lo siguiente:

- Si se digita la opción Tomar una vista previa, se muestra la imagen actual capturada por el lente de la cámara.
- Si se digita la opción Toma una imagen completa, se captura una foto. Buscar la foto capturada en la galería del dispositivo real.
- Si se digita el botón Mostrar una foto de la galería, se abre la ventana en la cual se selecciona una foto que se mostrará en la pantalla del dispositivo real.