CURSO DE TÉCNICO EN DESARROLLO NATIVO SOBRE PLATAFORMAS ANDROID

EOI – Escuela de Organización Industrial

TEMA 6. Recursos móviles

CONTENIDOS

1. Conectividad del dispositivo
2. Geolocalización y mapas
3. Cámara
4. Sensores
5. Contactos
6. Calendario

CONECTIVIDAD

- Gran porcentaje de las aplicaciones móviles requieren de una conexión a Internet para actualizar datos vía servidores
- Cada comunicación cliente-servidor resulta en un consumo de datos y batería
- Si el dispositivo no dispone de conectividad, tiene sentido realizar una comunicación cliente-servidor? No

CONECTIVIDAD

- **ConnectivityManager** nos permite determinar si el dispositivo dispone de conexión a Internet
- También podemos conocer el tipo de conexión: **WiFi, 3G, 4G**, etc…

doc:
https://developer.android.com/training/monitoring-device-state/connectivity-monitoring?hl=es

CONECTIVIDAD

- Cómo determinar si existe conexión a Internet
  - **NetworkInfo** contiene dicha información

```
ConnectivityManager cm = (ConnectivityManager)
    getSystemService(Context.CONNECTIVITY_SERVICE);

NetworkInfo activeNetwork = cm.getActiveNetworkInfo();
boolean isConnected = activeNetwork != null
    && activeNetwork.isConnectedOrConnecting();
```

CONECTIVIDAD

- **NetworkInfo**
  - **isConnectedOrConnecting()** devolverá **true** si el dispositivo tiene acceso a Internet, o **false** en caso contrario

```
ConnectivityManager cm = (ConnectivityManager)
    getSystemService(Context.CONNECTIVITY_SERVICE);

NetworkInfo activeNetwork = cm.getActiveNetworkInfo();
boolean isConnected = activeNetwork != null
    && activeNetwork.isConnectedOrConnecting();
```

CONECTIVIDAD

- Cómo determinar el tipo de conexión
  - **ConnectivityManager.TYPE_WIFI**
  - **ConnectivityManager.TYPE_MOBILE**
  - **ConnectivityManager.TYPE_ETHERNET**

```
ConnectivityManager cm = (ConnectivityManager)
   getSystemService(Context.CONNECTIVITY_SERVICE);

NetworkInfo activeNetwork = cm.getActiveNetworkInfo();
boolean isWifi = (activeNetwork.getType() ==
                    ConnectivityManager.TYPE_WIFI);
```

CONECTIVIDAD

- Para acceder al estado de la conectividad, es necesario declarar un permiso en **AndroidManifest**

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

PRÁCTICA

1. Detectar, en nuestra **ListActivity**, si el dispositivo dispone de conexión a Internet
2. Si dispone de conexión, realizar la comunicación cliente-servidor con Firebase
3. Si no, mostrar un **Toast** de error y no realizar la comunicación

CONECTIVIDAD

- La conectividad puede variar de un instante a otro en un dispositivo móvil
- Es una **buena práctica** comprobar siempre el estado de la conectividad antes de realizar una comunicación cliente-servidor, pero también podemos **registrarnos al sistema** para que nos notifique de cambios en la conectividad

CONECTIVIDAD

- Para **escuchar cambios de conectividad** debemos crear un **BroadcastReceiver** que reciba cambios de tipo…
  - `"android.net.conn.CONNECTIVITY_CHANGE"`
- … y declararlo en el fichero **AndroidManifest.xml**

CONECTIVIDAD

- **MyConnectivityBroadcastReceiver**
  - Hereda del componente Android **BroadcastReceiver**
  - El método **onReceive** se ejecutará cuando exista un cambio en la conectividad

```
public class MyConnectivityReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent) {
        // AQUI SE HA PRODUCIDO UN CAMBIO EN LA CONECTIVIDAD
    }
}
```

PRÁCTICA

1. Crear un BroadcastReceiver llamado **MyConnectivityBroadcastReceiver**
   - ***File > New > Other > Broadcast Receiver***
2. Comprobar en el método **onReceive** si el dispositivo está conectado a Internet e imprimir por **Log.d** la respuesta.

CONECTIVIDAD

- Un **BroadcastReceiver** por sí sólo no se registra en el sistema, por lo que debemos especificar mediante código que comience a escuchar cambios. Esta operación se conoce como ***registrar un receiver.***
- También deberemos dejar de escuchar cambios cuando no nos interesen más: ***dar de baja un receiver.***

CONECTIVIDAD

- Para **registrar un receiver:**
  - Creamos un objeto tipo **IntentFilter** con la acción que queremos escuchar
  - Utilizamos el método **registerReceiver** para indicar el componente y el intent filter a registrar

```
IntentFilter intentFilter = new IntentFilter();
intentFilter.addAction(ConnectivityManager
                                .CONNECTIVITY_ACTION);
registerReceiver(new MyConnectivityReceiver(), intentFilter);
```

PRÁCTICA

1. En el método **onCreate** de **ListActivity,** registrar **MyConnectivityBroadcastReceiver** para que escuche cambios en la conectividad del dispositivo
2. Alternar entre **modo avión / modo normal** y verificar que nuestro receiver está recibiendo los cambios de conectividad

GEOLOCALIZACIÓN

- Conocer la posición GPS del dispositivo puede ayudar a mostrar contenido de mayor interés para el usuario
- La ubicación del dispositivo se puede obtener a través de redes WiFi, redes móviles o satélites GPS

doc: https://developer.android.com/guide/topics/location/strategies

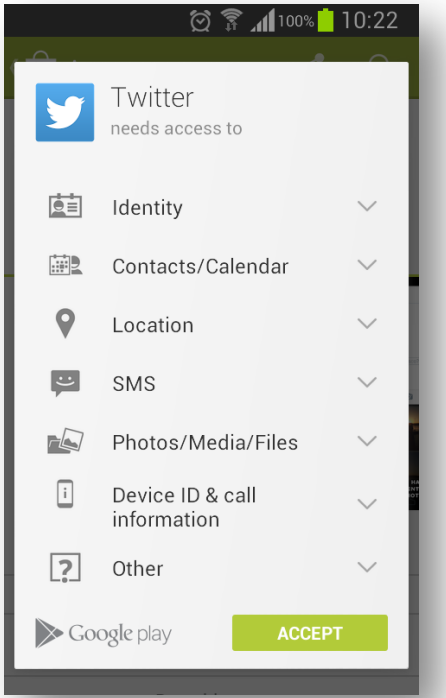
GEOLOCALIZACIÓN

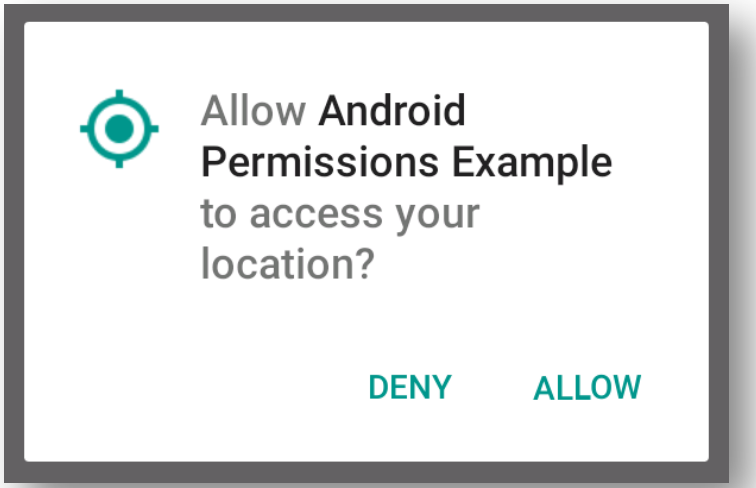
- La ubicación del dispositivo del usuario es una información muy sensible, por lo que Android nos obliga a pedir permisos **explícitamente**
  - Se definen en AndroidManifest.xml

```
<!-- Permisos para geolocalización sin GPS (menos precisa) ->
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-feature
android:name="android.hardware.location.network" />
<!-- Permisos para geolocalización con GPS (más precisa) -->
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-feature android:name="android.hardware.location.gps" />
```

GEOLOCALIZACIÓN

- A partir de **Android 6.0** necesitamos pedir dichos permisos en tiempo de ejecución, además de definirlos en **AndroidManifest.xml**





GEOLOCALIZACIÓN

- Pidiendo al usuario permiso en tiempo de ejecución:
  - Primero, debemos comprobar si ya disponemos de los permisos necesarios (por ejemplo, al arrancar la aplicación)

```
int permissionCheck = ContextCompat.checkSelfPermission(this,
        Manifest.permission.ACCESS_FINE_LOCATION);
if (permissionCheck == PackageManager.PERMISSION_GRANTED) {
    // Tenemos permisos
} else {
    // No tenemos permisos
}
```

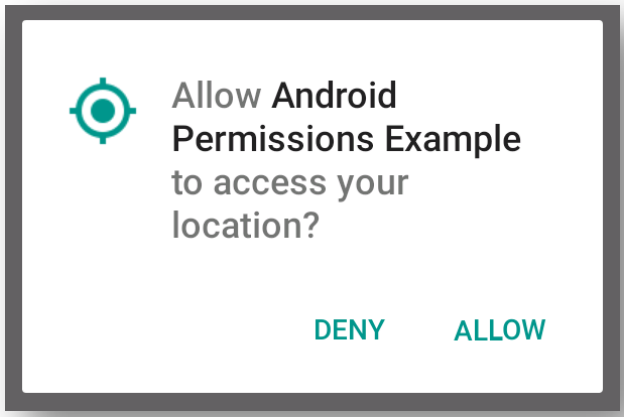
GEOLOCALIZACIÓN

- Pidiendo al usuario permiso en tiempo de ejecución:
  - Si no detectamos permisos, y nuestra aplicación los requiere, procedemos a solicitarlos

```
ActivityCompat.requestPermissions(this,
new String[] { Manifest.permission.ACCESS_FINE_LOCATION },
        100);
```

GEOLOCALIZACIÓN

- Pidiendo al usuario permiso en tiempo de ejecución:
  - Android mostrará un diálogo al usuario solicitando permisos
  - Tras la acción del usuario (aceptar/denegar) permisos, se ejecutará el método de nuestra Activity llamado **onRequestPermissionsResult**



GEOLOCALIZACIÓN

- **onRequestPermissionsResult**

```
@Override
public void onRequestPermissionsResult(int requestCode,
                                       String permissions[],
                                      int[] grantResults) {
    if (requestCode == 100) {
        if (grantResults.length > 0
         && grantResult[0] == PackageManager.PERMISSION_GRANTED)
        {
                // El usuario acepta los permisos
         } else{
                // El usuario deniega los permisos
         }
    }
}
```

PRÁCTICA

1. En el método **onCreate** de **ListActivity,** comprobar si el dispositivo dispone de permisos de geolocalización
2. Si no dispone, solicitarlos y mostrar al usuario el diálogo de petición de permisos

GEOLOCALIZACIÓN

- La obtención de la ubicación del dispositivo ocurre de manera **asíncrona**
- Pedimos a **LocationManager** que nos notifique de las actualizaciones de ubicación y definimos un *Listener* para recibir dichas actualizaciones
  - Para obtener el objeto **LocationManager**

```
LocationManager locationManager = (LocationManager)
    getSystemService(Context.LOCATION_SERVICE);
```

GEOLOCALIZACIÓN

- Métodos disponibles en **LocationManager:**
  - **getLastKnownLocation(String provider)** – Devuelve la última ubicación obtenida por el sistema de geolocalización
  - **requestLocationUpdates(String provider, long minTime, long minDistance, LocationListener listener)** – Empieza a escuchar cambios en la geolocalización del dispotivo

GEOLOCALIZACIÓN

- **LocationListener**
  - **onLocationChanged(Location location)** – Se ejecuta cuando obtenemos una nueva ubicación
  - **onStatusChanged(String provider, int status, Bundle extras)** – Cuando el estado del *provider* cambia
  - **onProviderEnabled(String provider)** – Se llama cuando el usuario activa el *provider*
  - **onProviderDisabled(String provider)** – Se llama cuando el usuario desactiva el *provider*

GEOLOCALIZACIÓN

- **LocationListener**

```
LocationListener listener = new LocationListener() {
    @Override
    public void onLocationChanged(Location location) {
        Log.d("Location Changed", location.toString());
    }

    @Override
    public void onStatusChanged(String provider, int status,
        Bundle extras) {}

    @Override
    public void onProviderEnabled(String provider) {}

    @Override
    public void onProviderDisabled(String provider) {}
};
```

GEOLOCALIZACIÓN

- **locationManager.requestLocationUpdates**
  - **String provider:** *provider* a utilizar (GPS_PROVIDER o NETWORK_PROVIDER)
  - **long minTime:** umbral de tiempo para recibir una nueva ubicación
  - **long minDistance:** umbral de distancia para recibir una nueva ubicación
  - **LocationListener listener:** *listener* para recibir las notificaciones de cambios

GEOLOCALIZACIÓN

- **LocationListener**

```
LocationManager locationManager = (LocationManager)
    getSystemService(Context.LOCATION_SERVICE);

LocationListener listener = new LocationListener() {
    @Override
    public void onLocationChanged(Location location) {
        Log.d("Location Changed",location.toString());
    }
    . . .
};

locationManager.requestLocationUpdates(
    LocationManager.GPS_PROVIDER,0,0,listener);
```

PRÁCTICA

1. Tras comprobar y verificar en **ListActivity** que disponemos de permisos de geolocalización, comenzar a escuchar los cambios e imprimir por pantalla la nueva ubicación cada vez que cambie

```
int permissionCheck = ContextCompat.checkSelfPermission(this,
        Manifest.permission.ACCESS_FINE_LOCATION);
if (permissionCheck == PackageManager.PERMISSION_GRANTED) {
    // Tenemos permisos. Registrar actualizaciones
}
```

CÁMARA

- **Android** ofrece una API para el uso de la cámara del dispositivo
  - Permite capturar fotografías
  - Permite capturar vídeos
  - Permite implementar nuestra propia interfaz para el uso de la cámara (requiere permisos)

CÁMARA

- Para poder utilizar la API de la cámara, primero debemos solicitar la funcionalidad en **AndroidManifest.xml**
  - **required = true** - un dispositivo sin cámara no podrá instalar la aplicación
  - **required = false** - un dispositivo sin cámara sí podrá instalar la aplicación

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

CÁMARA

- **Capturar fotografías**
  - La forma más rápida y sencilla de capturar una fotografía es **delegando la responsabilidad en otra aplicación**, normalmente la propia aplicación de cámara del dispositivo
  - Si existe una aplicación en el dispositivo que captura fotos, por qué debemos reinventar la rueda?
  - No requiere permisos de cámara en AndroidManifest.xml

CÁMARA

- **Capturar fotografías.** El proceso es el siguiente:
  1. Crear un Intent para abrir la aplicación de la cámara
  2. Crear un método para obtener la imagen capturada una vez se cierra la aplicación de la cámara

CÁMARA

- Creando el Intent y lanzándolo a Android para que actúe en consecuencia
  - **startActivityForResult** – método que lanzará el Intent y cuando la Activity destino se cierre, devolverá el foco a la primera Activity

```
Intent takePictureIntent = new
    Intent(MediaStore.ACTION_IMAGE_CAPTURE);

if (takePictureIntent.resolveActivity(getPackageManager())
                                                != null) {
    startActivityForResult(takePictureIntent, 100);
}
```

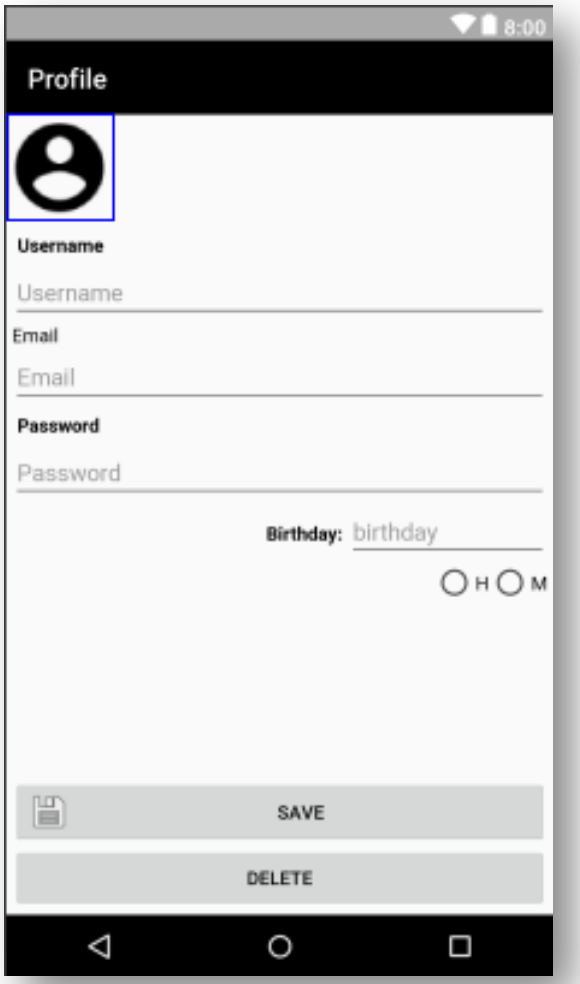
CÁMARA

- Una vez el usuario captura la imagen deseada, el foco regresa a la primera Activity ejecutando el método **onActivityResult**

```
@Override
protected void onActivityResult(int requestCode,
                                int resultCode, Intent data) {
    if (requestCode == 100 && resultCode == RESULT_OK) {
        Bundle extras = data.getExtras();
        Bitmap imageBitmap = (Bitmap) extras.get("data");
        mImageView.setImageBitmap(imageBitmap);
    }
}
```

PRÁCTICA

1. En **ProfileActivity**, cuando el usuario pulse la imagen de perfil:
    1. Crear un Intent para capturar una fotografía y lanzarlo
    2. Capturar el resultado con **onActivityResult** y mostrar la imagen capturada en la imagen de perfil



CÁMARA

- Si además de capturar la fotografía y mostrarla en un ImageView, queremos guardarla en disco:
  1. Debemos crear un fichero para almacenar la imagen
  2. Informar al Intent que queremos que la imagen se guarde en el fichero recién creado

CÁMARA

- Crear el fichero
  - El siguiente método crea un fichero en la **ruta de almacenamiento externo privada** de nuestra aplicación

```
private File createImageFile() {
    File storageDir =
    getExternalFilesDir(Environment.DIRECTORY_PICTURES);
    return new File(storageDir, "profile.jpg");
}
```

CÁMARA

- Crear el Intent y especificar el fichero

```
Intent takePictureIntent = new
    Intent(MediaStore.ACTION_IMAGE_CAPTURE);

if (takePictureIntent.resolveActivity(getPackageManager())
                                                != null) {
    File photoFile = createImageFile();
    Uri photoURI = FileProvider.getUriForFile(this,
                "com.android.teaching.miprimeraapp",
                 photoFile);
    takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT,
                                photoURI);
    startActivityForResult(takePictureIntent, 100);
}
```

CÁMARA

- Al pedir una aplicación externa que capture una imagen y la guarde en un fichero en la **ruta externa privada** de nuestra aplicación, necesitamos exponer dicha ruta con un **FileProvider**
- **FileProvider** expondrá de forma pública (para aquellas aplicaciones interesadas), el contenido interno del directorio externo privado

CÁMARA

- Para crear un sencillo **FileProvider** a modo de ejemplo:
  1. Declararlo en AndroidManifest.xml

```
<provider
    android:name="android.support.v4.content.FileProvider”
    android:authorities="com.android.teaching.miprimeraapp”
    android:exported=“false”
    android:grantUriPermissions=“true”>
    <meta-data
        android:name=“android.support.FILE_PROVIDER_PATHS”
        android:resource=“@xml/file_paths” />
</provider>
```

CÁMARA

- Para crear un sencillo **FileProvider** a modo de ejemplo:
  2. Crear el fichero **@xml/file_paths**, almacenado en la carpeta **res/xml** del proyecto

```
<?xml version="1.0" encoding="utf-8"?>
<paths xmlns:android="http://schemas.android.com/apk/res/
android">
    <external-path name="my_images"
        path="Android/data/com.android.teaching.miprimeraapp/
                files/Pictures" />
</paths>
```

PRÁCTICA

1. Eliminar de **ProfileActivity** el método **onActivityResult**
2. Modificar el Intent de hacer una fotografía solicitando que la imagen se guarde en un fichero de la ruta privada externa de nuestra aplicación
3. En el método **onResume** de **ProfileActivity** cargar en el ImageView de perfil la imagen del fichero

CÁMARA

- Para capturar un vídeo, el proceso es idéntico, solo que cambian las constantes a utilizar:
  1. El Intent se debe crear con **MediaStore.ACTION_VIDEO_CAPTURE**
  2. En el método **onActivityResult,** el objeto **intent.getData()** representa directamente la Uri del video guardado

CÁMARA

- Para capturar un vídeo

```
// 1) Crear Intent
Intent takeVideoIntent = new
    Intent(MediaStore.ACTION_VIDEO_CAPTURE);
startActivityForResult(takeVideoIntent, 101);

// 2) Obtener la Uri del video
@Override
protected void onActivityResult(int requestCode,
                                int resultCode, Intent intent) {
    if (requestCode == 101 && resultCode == RESULT_OK) {
        Uri videoUri = intent.getData();
        mVideoView.setVideoURI(videoUri);
    }
}
```

SENSORES

- Android ofrece diferentes APIs para monitorear el movimiento de un dispositivo:
  - La gravedad
  - Aceleración lineal
  - Rotación
  - Contador de pasos
  - Acelerómetro
  - Giroscopio
  - …

doc: https://developer.android.com/guide/topics/sensors/sensors_motion

SENSORES

- Cada uno de estos sensores se obtiene de un objeto llamado **SensorManager** a través del método **getDefaultSensor(int type)**
  - Por ejemplo, el acelerómetro:

```
SensorManager sensorManager = (SensorManager)
    getSystemService(Context.SENSOR_SERVICE);
Sensor mySensor = sensorManager
    .getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

SENSORES

- Tipos de sensores que podemos obtener:
  - Sensor.TYPE_ACCELEROMETER
  - Sensor.TYPE_GRAVITY
  - Sensor.TYPE_GYROSCOPE
  - Sensor.TYPE_ROTATION_VECTOR
  - Sensor.TYPE_STEP_COUNTER
  - Sensort.TYPE_PROXIMITY
  - …

SENSORES

- La existencia de dichos sensores depende, muchas veces, del dispositivo, por lo que siempre hay que comprobar que el sensor existe:

```
if(mySensor != null) {
     // Escuchar las actualizaciones del sensor
}
```

SENSORES

- ¿Cómo escuchar las actualizaciones de cualquier sensor?
  - Implementando la interfaz **SensorEventListener**

```
sensorManager.registerListener(new SensorEventListener() {
    @Override
    public void onSensorChanged(SensorEvent event) {
        Log.d("Sensors",
                "Accelerometer: " +event.values.toString());
    }

    @Override
    public void onAccuracyChanged(Sensor sensor,
            int accuracy) { }
}, mySensor, SensorManager.SENSOR_DELAY_NORMAL);
```

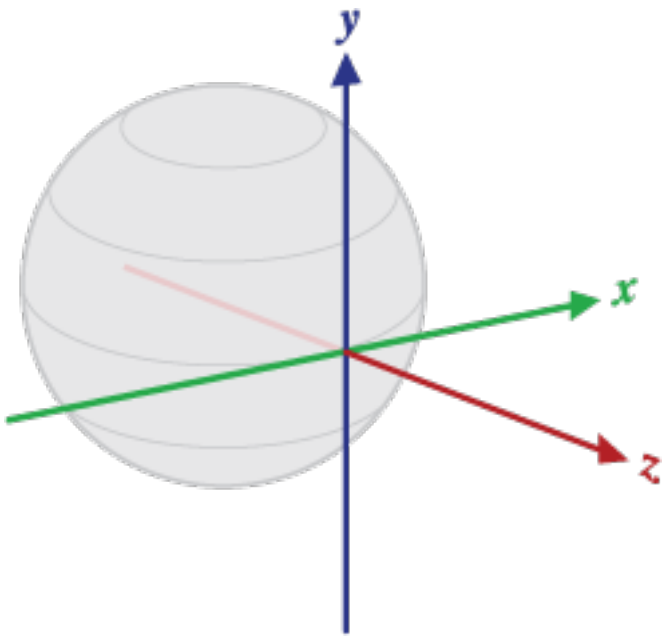
SENSORES

- Siempre que se registra un Listener para un sensor, hay que liberarlo en el método **onStop** u **onDestroy**

```
@Override
protected void onStop() {
    super.onStop();
    SensorManager sensorManager = (SensorManager)
                    getSystemService(Context.SENSOR_SERVICE);
    sensorManager.unregisterListener(myListener);
}
```

SENSORES

- Los sensores devuelven valores (normalmente de tipo float) que representan variaciones en alguno de los ejes de coordenadas del dispositivo

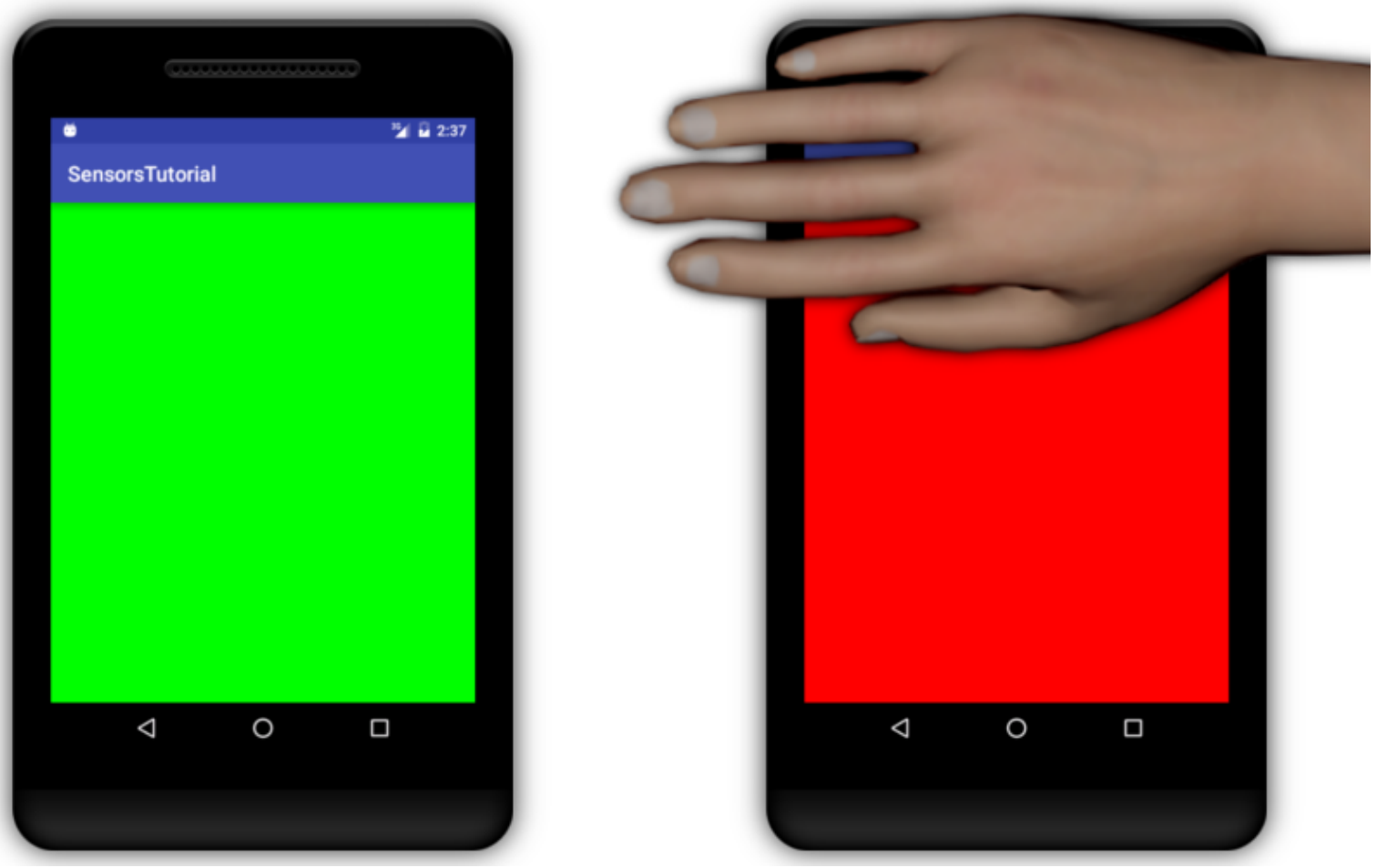


PRÁCTICA

1. Crear una Activity nueva llamada **SensorsActivity** que se abra con un nuevo botón en la toolbar:
    1. En **onResume** registre un listener para el sensor **Sensor.TYPE_PROXIMITY**
    2. En el método **onSensorChanged** del listener cambiar el color de fondo de la pantalla

```
@Override
public void onSensorChanged(SensorEvent event) {
    if(event.values[0] < mySensor.getMaximumRange()) {
        getWindow().getDecorView()
            .setBackgroundColor(Color.RED);
    } else{
        getWindow().getDecorView()
            .setBackgroundColor(Color.GREEN);
    }
}
```

PRÁCTICA



PRÁCTICA

1. En el método **onResume** de **SensorsActivity:**
    1. Registrar un nuevo listener para el sensor Sensor.TYPE_ACCELEROMETER
    2. En el método **onSensorChanged**, imprimir los valores de los ejes en **Log.d**: **event.values[0]**, **event.values[1]** y **event.values[2]**

PRÁCTICA

1. En el método **onStop** de **SensorsActivity**
    1. Liberar los listeners registrados de todos los sensores

```
@Override
protected void onStop() {
    super.onStop();
    SensorManager sensorManager = (SensorManager)
                    getSystemService(Context.SENSOR_SERVICE);
    sensorManager.unregisterListener(myListener);
}
```