



# Desarrollo de Aplicaciones para Android

## Sesión 4: Eventos y sensores



# Puntos a tratar

- Entrada en dispositivos móviles
- Pantalla táctil, gestos, y *multitouch*
- Orientación y aceleración
- Geolocalización
- Reconocimiento del habla

# Entrada en dispositivos móviles

- No hay teclado y ratón, pero ...
- Si que tenemos:
  - Pantalla táctil
  - *Multitouch*
  - Acelerómetro
  - Giroscopio
  - Brújula
  - GPS y red celular
  - Micrófono
  - Cámara





# Pantalla táctil

- Principal forma de entrada en móviles Android
- *Gesto*
  - Comienza al poner un dedo en la pantalla
  - Continúa mientras el dedo se mueve
  - Termina al levantarlo
- Captura de eventos *touch*
  - Implementar un objeto `OnTouchListener`
  - Sobrescribir `onTouchEvent` de `View`



# Evento de *touch*

- Recibimos datos del evento en `MotionEvent`
- Devolvemos
  - `true` para seguir recibiendo eventos del gesto
  - `false` en caso contrario

```
public class MiComponente extends View
    ...
    @Override
    public boolean onTouchEvent(MotionEvent event) {
        // Procesar evento
        return true;
    }
    ...
}
```



# Tipos de eventos *touch*

- Con `getAction` de `MotionEvent`
  - `ACTION_DOWN`
    - Comienzo del gesto. Se pone el dedo en la pantalla.
  - `ACTION_MOVE`
    - Continuación del gesto. El dedo se mueve a otra posición de la pantalla.
  - `ACTION_UP`
    - Fin del gesto. Se levanta el dedo de la pantalla.
  - `ACTION_CANCEL`
    - Fin del gesto. Otro componente toma el control.
- Obtenemos coordenadas con `getX` y `getY`

# Ejemplo sencillo

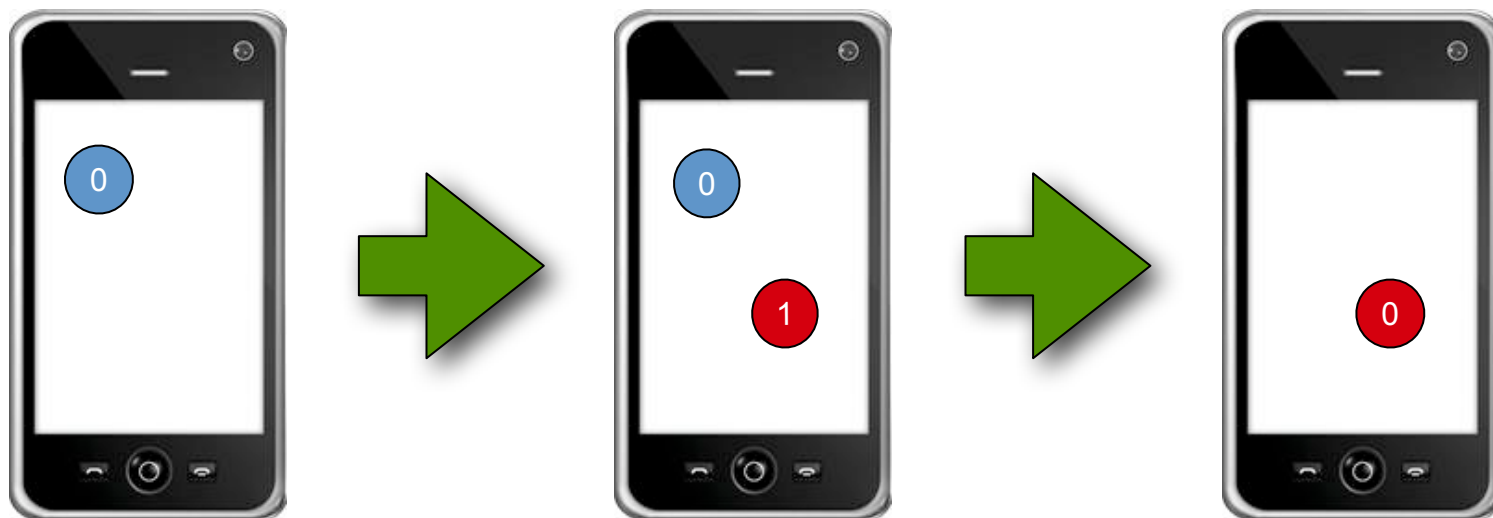
```
@Override
public boolean onTouchEvent(MotionEvent event) {
    if(event.getAction() == MotionEvent.ACTION_MOVE) {
        x = event.getX();
        y = event.getY();

        this.invalidate();
    }
    return true;
}
```

Fuerza a repintar  
el componente

# Multitouch

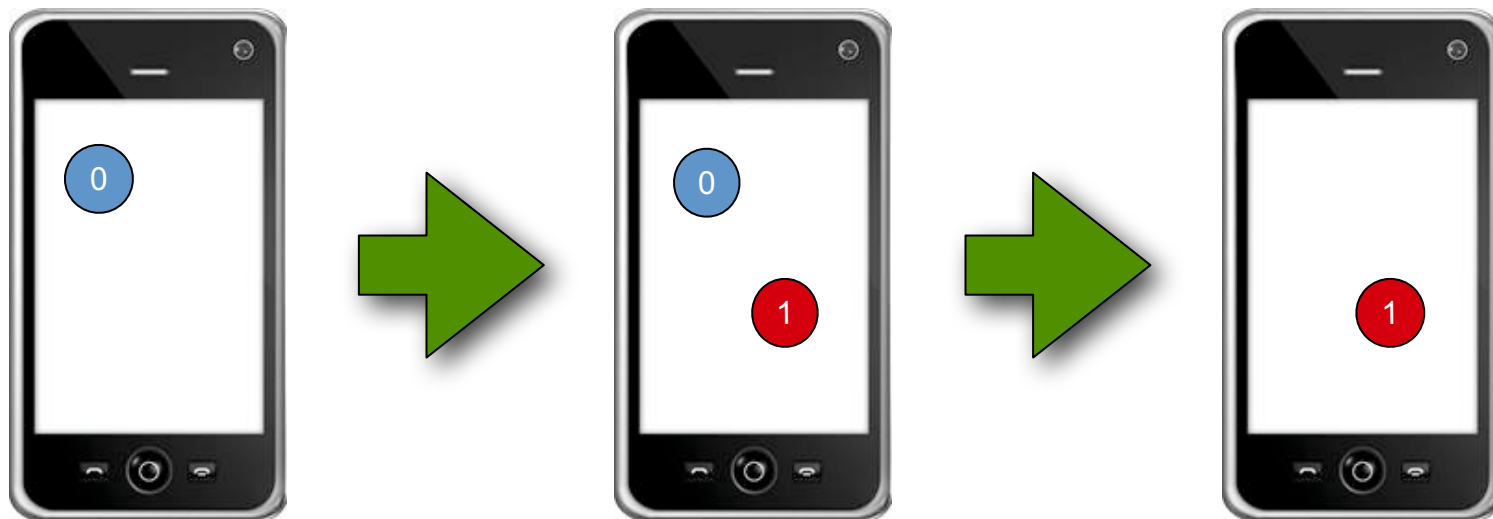
- `MotionEvent` contiene un *array* de punteros
  - Índices de 0 a `getPointerCount`
  - Posición de un puntero
    - `getX(indice)`, `getY(indice)`
  - Los índices pueden cambiar





# Identificadores

- Cada puntero tiene un identificador
  - El identificador es propio de cada gesto
  - Obtener el identificador de un índice
    - `getPointerId(indice)`
  - Buscar un puntero dado su identificador
    - `findPointerIndex(id)`





# Punteros secundarios

- Nuevos tipos de eventos
  - ACTION\_POINTER\_DOWN
    - Un puntero se pone en la pantalla habiendo ya otro.
  - ACTION\_POINTER\_UP
    - Un puntero se quita de la pantalla quedando otro en ella.
- Separar acción e índice del evento

```
final int accion = event.getAction() &
                    MotionEvent.ACTION_MASK;
final int indice = (event.getAction() &
                    MotionEvent.ACTION_POINTER_INDEX_MASK)
                    >> MotionEvent.ACTION_POINTER_INDEX_SHIFT;
```



# Reconocimiento de gestos

- *Gesture detectors*
  - Clases que encapsulan reconocimiento de gestos
  - Nos permiten reconocer gestos de alto nivel
- `GestureDetector` **reconoce**
  - *Single tap*
  - *Double tap*
  - *Scroll*
  - *Fling* (lanzamiento)
  - *Mantener*
- `ScaleGestureDetector` **reconoce** *pinch*

# Gestos simples (I)

- *Listener* con eventos de alto nivel

```
class ListenerGestos extends
    GestureDetector.SimpleOnGestureListener {
    @Override
    public boolean onDown(MotionEvent e) {
        return true;
    }

    @Override
    public boolean onDoubleTap(MotionEvent e) {
        // Tratar el evento
        return true;
    }
}
```

¡Muy importante! Para seguir procesando el gesto

## Gestos simples (II)

- Llamamos al detector en `onTouchEvent`

```
GestureDetector detectorGestos;

public ViewGestos(Context context) {
    super(context);

    ListenerGestos lg = new ListenerGestos();
    detectorGestos = new GestureDetector(lg);
    detectorGestos.setOnDoubleTapListener(lg);
}

@Override
public boolean onTouchEvent(MotionEvent event) {
    return detectorGestos.onTouchEvent(event);
}
```

# Sensores

- Accesibles mediante `SensorManager`
  - Aceleración
  - Orientación
  - Brújula
  - Giroscopio
  - Luz
  - Proximidad
  - Temperatura
  - Presión
- Se representan con `Sensor`

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



# Listener de sensores

```
class ListenerSensor implements SensorEventListener {  
  
    public void onSensorChanged(SensorEvent sensorEvent) {  
  
        // La lectura del sensor ha cambiado  
        float [] lecturas = sensorEvent.values;  
  
        // Las lecturas dependen del tipo de sensor  
        ...  
    }  
  
    public void onAccuracyChanged(Sensor sensor,  
                                   int accuracy) {  
        // La precisión del sensor ha cambiado  
    }  
}
```

# Lecturas del sensor

- Comenzar las lecturas

```
ListenerSensor listener = new ListenerSensor();  
sensorManager.registerListener(listener,  
    sensor, SensorManager.SENSOR_DELAY_NORMAL);
```



SENSOR_DELAY_FASTER	SENSOR_DELAY_NORMAL
SENSOR_DELAY_GAME	SENSOR_DELAY_UI

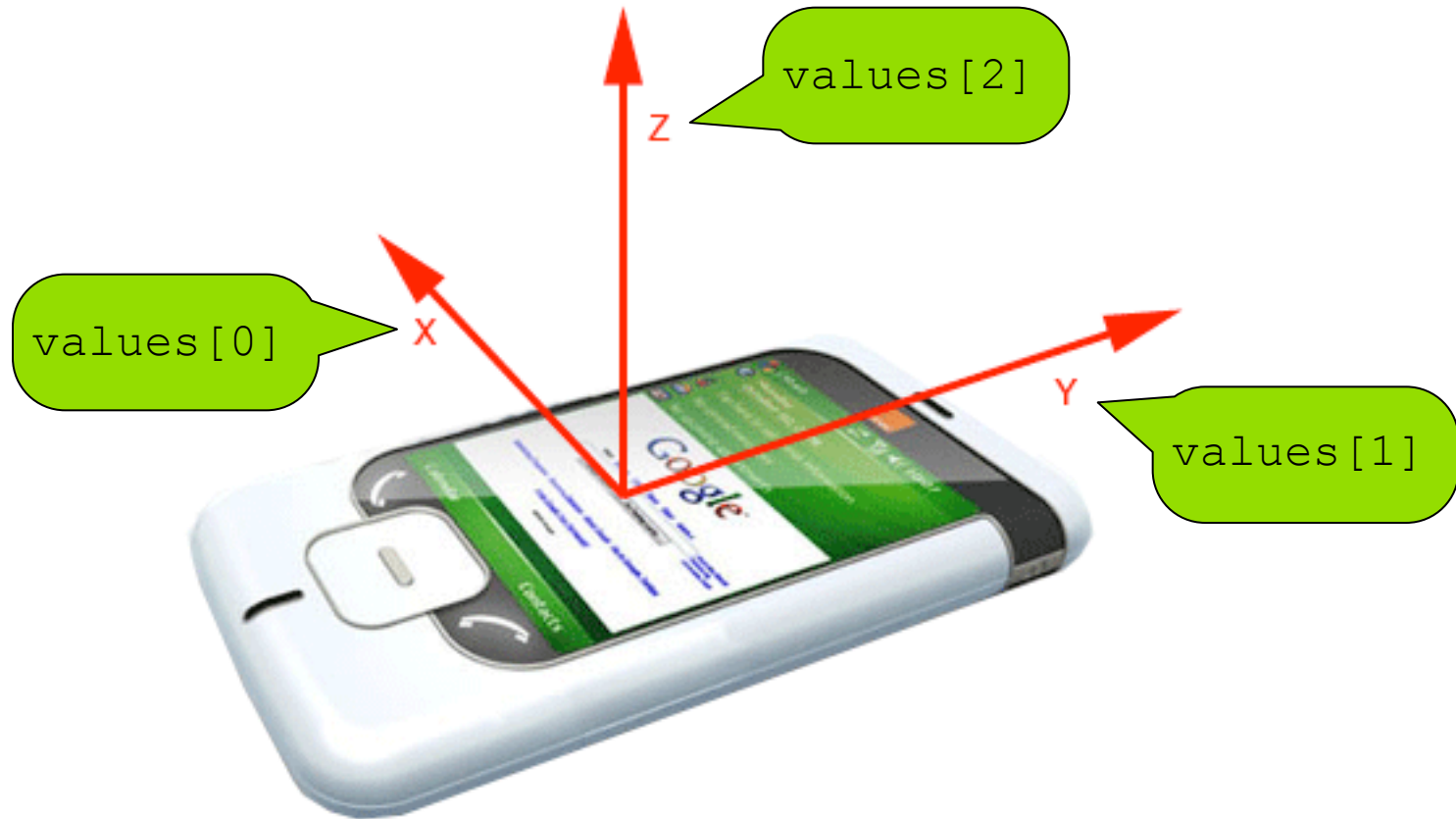
- Detener las lecturas (*IMPORTANTE*)

```
sensorManager.unregisterListener(listener);
```

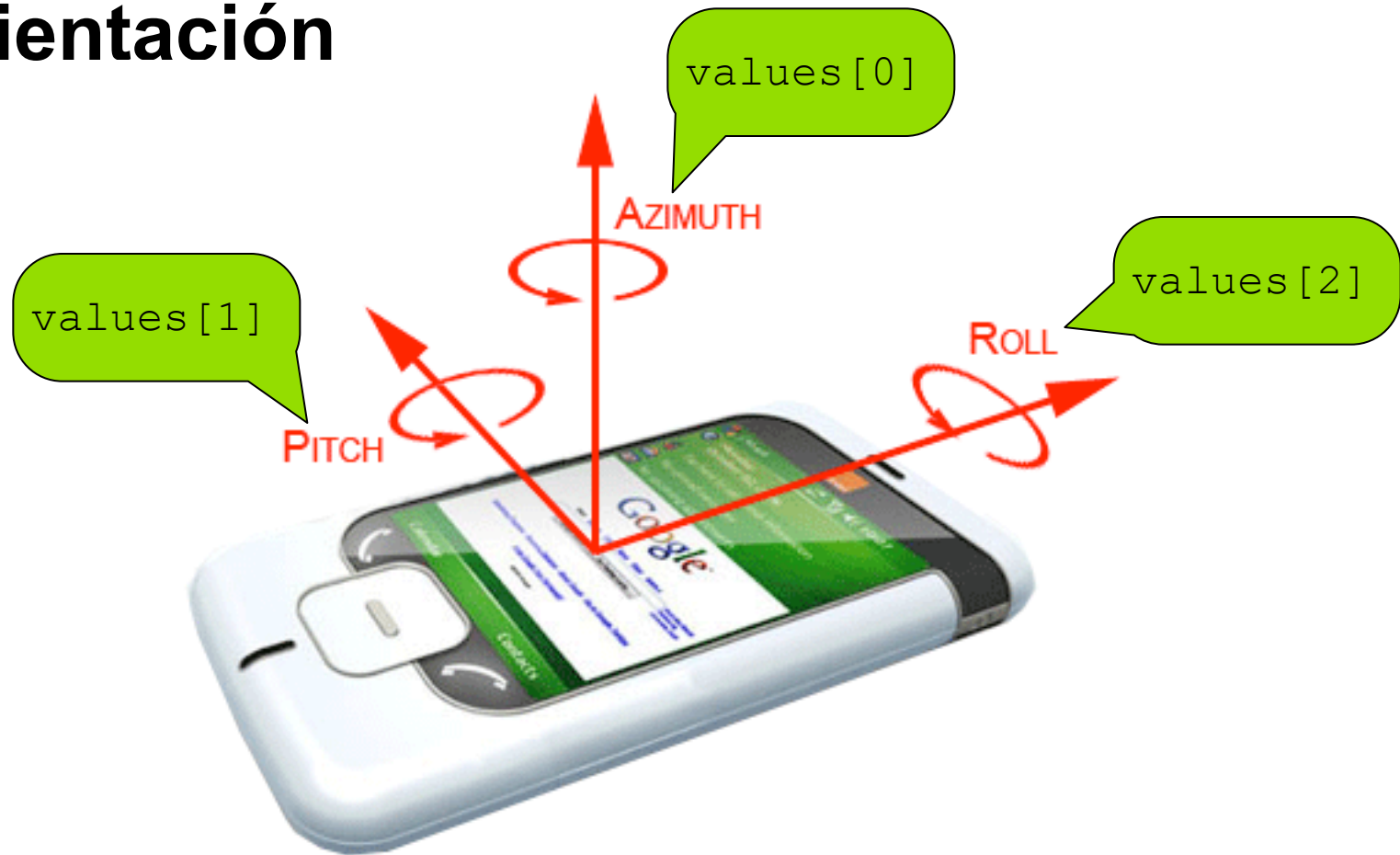




# Aceleración



# Orientación





# Combinar sensores

- El sensor `ORIENTATION` está desaprobado
- Podemos mejorar la orientación combinando
  - Acelerómetro
  - Brújula

```
float[] values = new float[3];  
float[] R = new float[9];  
SensorManager.getRotationMatrix(R, null,  
    valuesAcelerometro, valuesBrujula);  
SensorManager.getOrientation(R, values);
```



# Geolocalización

- Proveedores

- GPS

- Localización fina
    - Necesita dispositivo GPS

- Red

- Localización aproximada
    - Usa la célula de la red móvil

- Permisos

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



# Obtener última localización

- Obtiene última posición registrada
- No solicita actualizarla al proveedor

```
LocationManager manager = (LocationManager)  
    this.getSystemService(Context.LOCATION_SERVICE);  
  
Location posicion = manager  
    .getLastKnownLocation(LocationManager.GPS_PROVIDER);
```

- El objeto `Location` proporciona
  - Latitud
  - Longitud
  - Altura
  - Velocidad
  - etc ...



# Obtener nueva posición

```
class ListenerPosicion implements LocationListener {  
  
    public void onLocationChanged(Location location) {  
        // Recibe nueva posición.  
    }  
    public void onProviderDisabled(String provider){  
        // El proveedor ha sido desconectado.  
    }  
    public void onProviderEnabled(String provider){  
        // El proveedor ha sido conectado.  
    }  
    public void onStatusChanged(String provider,  
                                int status, Bundle extras){  
        // Cambio en el estado del proveedor.  
    }  
};
```



# Solicitar actualización de posición

- Registramos el *listener*

```
ListenerPosicion listener = new ListenerPosicion();  
long tiempo = 5000; // 5 segundos  
float distancia = 10; // 10 metros  
  
manager.requestLocationUpdates(  
    LocationManager.GPS_PROVIDER,  
    tiempo, distancia, listenerPosicion);
```

*Puede tardar en obtener una primera posición*

- Detenemos las actualizaciones

```
manager.removeUpdates(listener);
```



# Alertas de proximidad

- Definimos un receptor de intents

```
public class ReceptorProximidad extends BroadcastReceiver {  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        String key = LocationManager.KEY_PROXIMITY_ENTERING;  
        Boolean entra = intent.getBooleanExtra(key, false);  
        ...  
    }  
}
```

- Programamos el aviso

```
Intent intent = new Intent(codigo);  
PendingIntent pi = PendingIntent.getBroadcast(this, -1, intent, 0);  
manager.addProximityAlert(latitud, longitud, radio,  
                           caducidad, pi);  
  
IntentFilter filtro = new IntentFilter(codigo);  
registerReceiver(new ReceptorProximidad(), filtro);
```

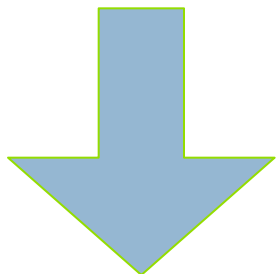


# Geocoder

- Transforma entre coordenadas y dirección

## Directo

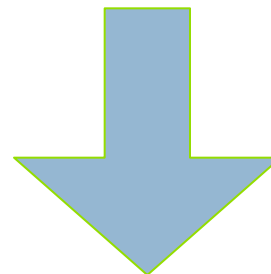
“Universidad de Alicante”



[38.3852333,-0.51515]

## Inverso

[38.3852333,-0.51515]



“Universidad de Alicante”

# Uso del geocoder

- Obtener el objeto `Geocoder`

```
Geocoder geocoder = new Geocoder(this,  
                                   Locale.getDefault());
```

- Transformación directa

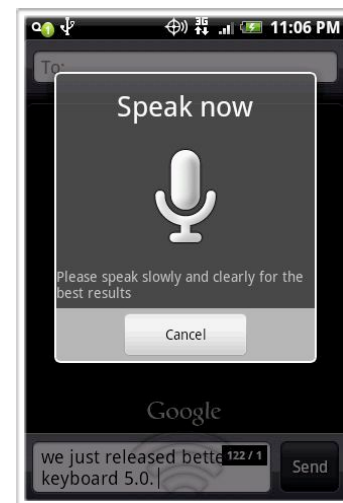
```
List<Address> coordenadas = geocoder  
    .getFromLocationName(direccion, maxResults);
```

- Transformación inversa

```
List<Address> direcciones = geocoder  
    .getFromLocation(latitud, longitud, maxResults);
```

# Reconocimiento del habla

- Transforma nuestra voz en texto
- Soporta diferentes idiomas
  - Propiedad `EXTRA_LANGUAGE`
  - Por ejemplo `"es-ES"`
- Dos modelos de lenguaje:
  - Búsqueda web (`LANGUAGE_MODEL_WEB_SEARCH`)
  - Libre (`LANGUAGE_MODEL_FREE_FORM`)
- Indicar el modelo de lenguaje es obligatorio
  - Propiedad `EXTRA_LANGUAGE_MODEL`



# Lanzar reconocimiento del habla

- Lanzamos la aplicación

```
Intent intent = new Intent(  
    RecognizerIntent.ACTION_RECOGNIZE_SPEECH);  
  
intent.putExtra(parametro, valor);  
  
startActivityForResult(intent, codigo);
```

Propiedades del  
reconocimiento  
(modo, idioma, etc)

- Obtenemos resultados

```
@Override  
protected void onActivityResult(int requestCode,  
    int resultCode, Intent data) {  
    if (requestCode == codigo && resultCode == RESULT_OK) {  
        ArrayList<String> resultados =  
            data.getStringArrayListExtra(  
                RecognizerIntent.EXTRA_RESULTS);  
    }  
    super.onActivityResult(requestCode, resultCode, data);  
}
```



# ¿Preguntas...?