Computación Móvil
Permisos, acceso a hardware interno

Pontificia Universidad Javeriana
Ingeniería de Sistemas
Profesor: Carlos Andrés Parra
E-mail: ca.parraa@javeriana.edu.co

Quiz!!

- ¿Para qué se usa un FrameLayout?
- ¿Para qué sirve un Adapter?
- ¿Qué clase contiene los identificadores de las vistas definidas en los layouts?
- ¿Qué atributo se debe definir para cargar los valores de un spinner?

Sesión 6

Android

Permisos, Acceso a Datos y Cámara

Permisos

- En Android, cada aplicación corre de forma separada de las demás, dentro de un SandBox.
- Cada aplicación debe pedir explícitamente permisos para acceder a los recursos que necesita fuera de su sandbox a través de "*permissions*"
- Dependiendo de lo sensible del recurso, Android puede dar el permiso de forma automática o solicitar el permiso al usuario final.
- Una aplicación no tiene permisos por defecto.

Permisos

- Hasta Android 5.1 API 22
  - Se solicitan los permisos al momento de instalar la aplicación, o en las actualizaciones. Una vez la aplicación está instalada, la única forma de quitar el permiso es desinstalándola.
  - Permisos en grupo, o se autoriza todo o no se instala.
- Desde de Android 6 Api 23
  - La aplicación pide permiso al usuario en ejecución. El usuario puede quitar los permisos en cualquier momento. De esta forma, la aplicación debe verificar que aún tenga permiso para acceder al recurso que necesita cada vez que se ejecuta!
  - La instalación se hace más simple, pero cada permiso se debe manejar de forma individual y en ejecución. Una aplicación puede tener algunos permisos rechazados y otros aceptados.

Permisos
Normales vs. Con Riesgo

- ***Normales***
  - Se necesitan datos o recursos por fuera del *sandbox* pero hay muy poco riesgo para la privacidad del usuario o para el buen funcionamiento de Android
    - *Internet, bluetooth, NFC, alarmas, zona horaria, vibración, wallpaper, audio, etc.*
    - *https://developer.android.com/guide/topics/permissions/normal-permissions.html*
  - **Android lo autoriza automáticamente!**

Permisos
Normales vs Con Riesgo

- ***Con riesgo!***
  - Se necesitan datos o recursos por fuera del *sandbox* que involucran información privada del usuario, modifican la información almacenada, o la operación de otras aplicaciones:
    - *Calendario, cámara, contactos, localización fina y gruesa, grabar audio, registro de llamadas, SMS, leer y escribir en el almacenamiento externo:*
    - *https://developer.android.com/guide/topics/permissions/requesting.html#normal-dangerous*
  - **El usuario tiene que dar el permiso de forma explícita**

¿Cómo solicitar el permiso? Hasta Android 5.1 API 22

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.android.app.myapp" >
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    ...
</manifest>
```

- Internet
  - `<uses-permission android:name="android.permission.INTERNET”/>`
- Acceso a la cámara
  - `<uses-permission android:name="android.permission.CAMERA”/>`
- Acceso al almacenamiento externo
  - Lectura
    - ✓ `<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE”/>`
  - Escritura
    - ✓ `<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE”/>`

https://developer.android.com/reference/android/Manifest.permission.html

¿Cómo solicitar el permiso? Desde Android 6 API 23

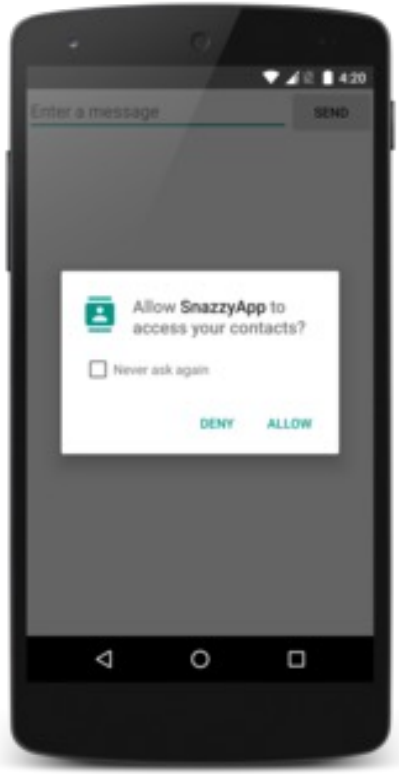
- Si es un permiso con riesgo, y el API es 23 o posterior, además de definir el permiso en el *Manifest* de la aplicación, es necesario verificar en ejecución el permiso y eventualmente solicitarlo usando el método ContextCompat.checkSelfPermission():

```
// Assume thisActivity is the current activity
int permissionCheck =
ContextCompat.checkSelfPermission(thisActivity,
        Manifest.permission.WRITE_CALENDAR);
```

- Este método retorna:
  - Si hay permiso: PackageManager.PERMISSION_GRANTED
  - Si no hay permiso: PackageManager.PERMISSION_DENIED

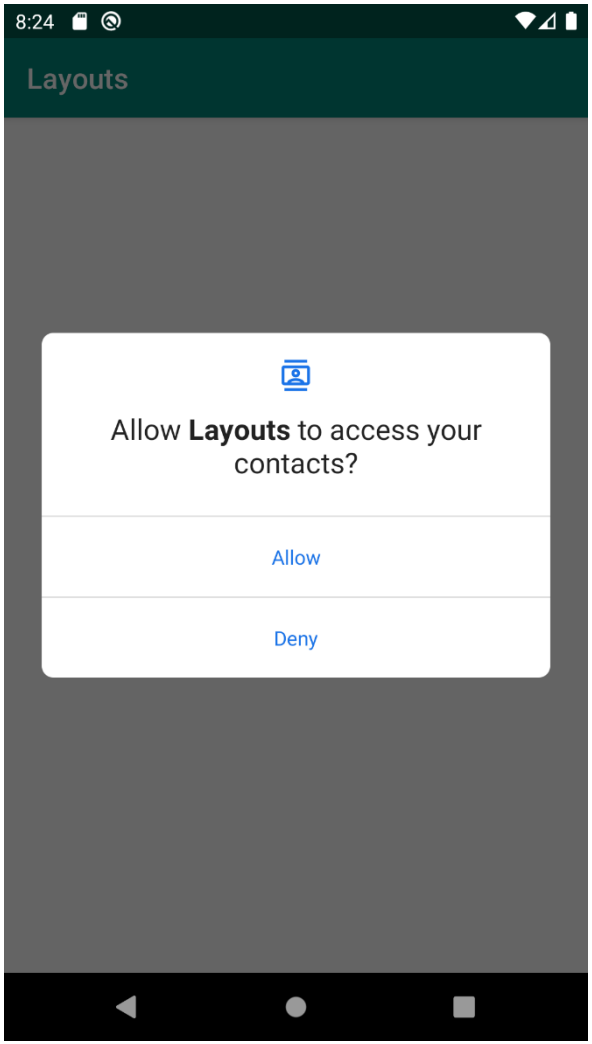
¿Cómo solicitar el permiso? Dar una justificación

- Se puede dar una justificación corta de tal manera que se explique al usuario por qué la aplicación esta solicitando el permiso.
- La estrategia recomendada es mostrar esta justificación sólo si el usuario rechaza el permiso solicitado, pero sigue intentando utilizar una funcionalidad que requiere de datos o recursos asociados a un permiso con riesgo.
- Android proporciona un método de utilidad:
  - shouldShowRequestPermissionRationale() retorna true si la app ha solicitado el permiso anteriormente y el usuario ha rechazado la solicitud.

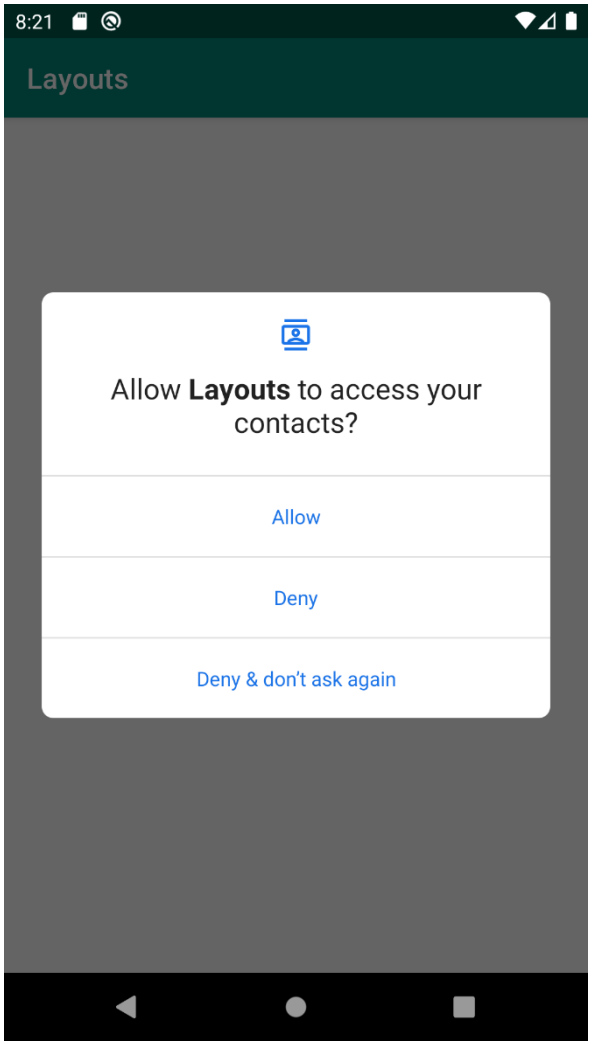


Nota. Si el usuario selecciona la opción *"Never ask again"* el método siempre retornará falso.

En las versiones recientes de Android



primera vez



Varios intentos

Solicitar un permiso

```
// Here, thisActivity is the current activity
if (ContextCompat.checkSelfPermission(thisActivity,
                Manifest.permission.READ_CONTACTS)
        != PackageManager.PERMISSION_GRANTED) {
    // Should we show an explanation?
    if
(ActivityCompat.shouldShowRequestPermissionRationale(thisActivity,
            Manifest.permission.READ_CONTACTS)) {

        // Show an expanation to the user *asynchronously*
    }
    // request the permission.
    ActivityCompat.requestPermissions(thisActivity,
              new String[]{Manifest.permission.READ_CONTACTS},
              MY_PERMISSIONS_REQUEST_READ_CONTACTS);
        // MY_PERMISSIONS_REQUEST_READ_CONTACTS es una
        // constante definida en la aplicación, se debe usar
        // en el callback para identificar el permiso
    }
}
```

Solicitar Permiso

- Idealmente se puede dejar el código en un método genérico qué reciba los cuatro parámetros necesarios: el contexto (la actividad), la cadena del permiso a solicitar, la justificación (si se quiere mostrar, y el código que identifica la solicitud).
- Este método se puede reutilizar para todos los permisos que requieran las diferentes actividades de la aplicación.

```
private void requestPermission(Activity context,
            String permiso, String justificacion, int idCode){
    // El contexto es la actividad principal
    if (ContextCompat.checkSelfPermission(context, permiso)
!=              PackageManager.PERMISSION_GRANTED) {
        //…
}
```

Solicitar Permiso
Proceso completo

- El sistema muestra el diálogo estándar (no se puede modificar) y cuando el usuario responde, el sistema llama al callback onRequestPermissionResult():

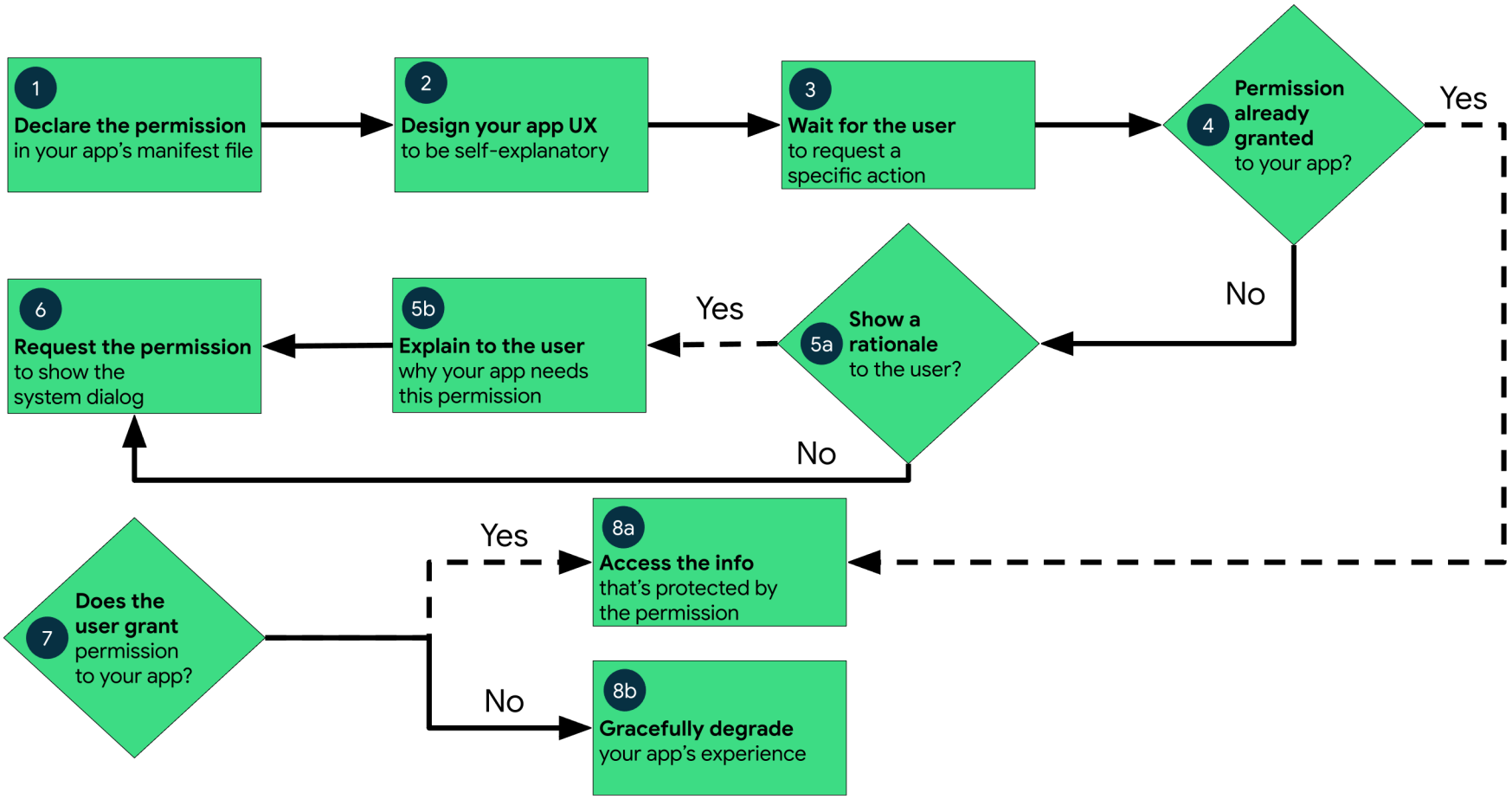
```
@Override
public void onRequestPermissionsResult(int requestCode, String permissions[], int[] grantResults)
{
    switch (requestCode) {
        case MY_PERMISSIONS_REQUEST_READ_CONTACTS: {
            // If request is cancelled, the result arrays are empty.
            if (grantResults.length > 0
                && grantResults[0] == PackageManager.PERMISSION_GRANTED) {

                // permission was granted, continue with task related to permission

            } else {

                // permission denied, disable functionality that depends on this permission.
            }
            return;
        }
        // other 'case' lines to check for other
        // permissions this app might request
    }
}
```

Grupos de Permisos

- Todos los permisos pertenecen a grupos.
- Cuando se solicita un permiso a través del diálogo, el sistema describe el grupo completo de permisos.
- Si un permiso con riesgo ya se aceptó y existe otro del mismo grupo en el ***manifest***, el sistema otorga el segundo permiso sin interacción con el usuario.
- Por ejemplo, si a una app ya se le otorgó el permiso READ_CONTACTS y luego esta solicita el permiso WRITE_CONTACTS, el sistema lo otorga de inmediato.
- La lista de grupos de permisos se puede consultar en:

https://developer.android.com/guide/topics/security/permissions.html#perm-groups

Permission Workflow



https://developer.android.com/training/permissions/requesting#java

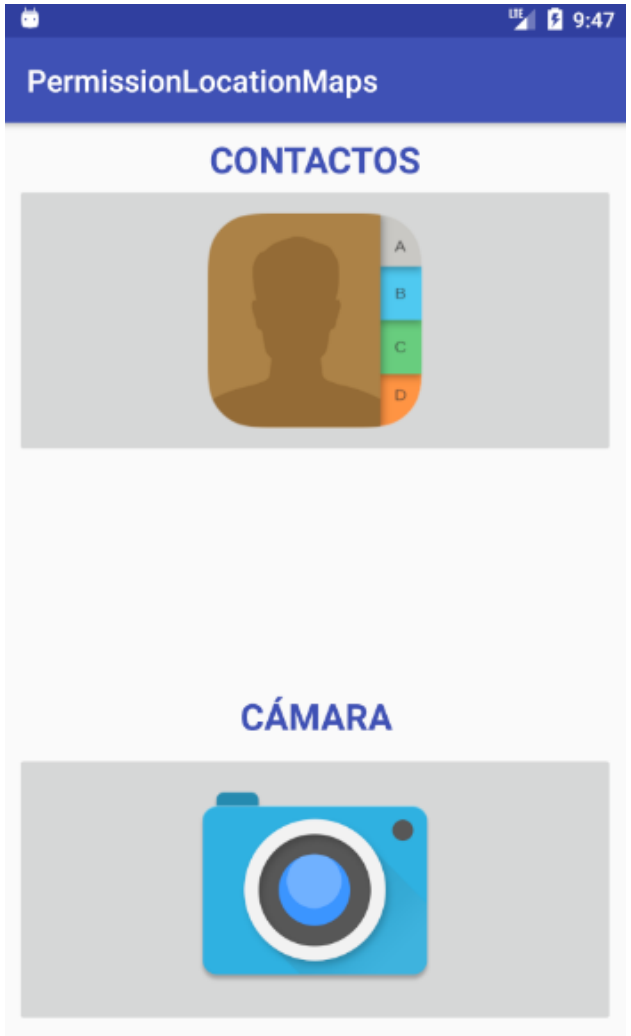
Activity Resolution API
Request a permission

```
ActivityResultLauncher<String> getSinglePermission = registerForActivityResult(
        new ActivityResultContracts.RequestPermission(),
        new ActivityResultCallback<Boolean>() {
            @Override
            public void onActivityResult(Boolean result) {
                if(result == true){
                    //granted
                }else{
                    //denied
                }
            }
});
```

```
//onCreate
getSinglePermission.launch(Manifest.permission.READ_CONTACTS);
```

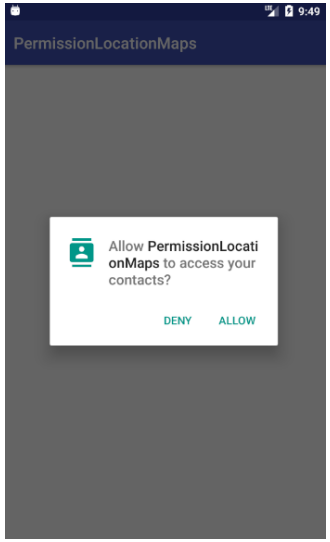
Ejercicio 1

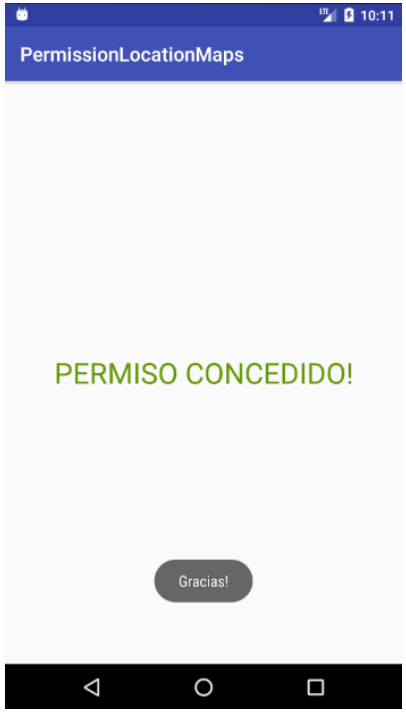
- Defina una actividad principal con el layout de la figura utilizando un ConstraintLayout.



Ejercicio 1 (Cont)

- Programe el botón Contactos para que se lance una actividad que verifique y, en caso de ser necesario, solicite el permiso para leer los contactos del dispositivo.
- Muestre un Toast diciendo si el permiso ha sido concedido o denegado. En este último caso, proporcione una justificación para el permiso.
- Una vez el permiso sea concedido, muestre un TextView dónde se informe al usuario el estado del permiso como se muestra en la figura.





Acceso a los contactos del teléfono

- Primero se define el permiso:
  - `<uses-permission android:name="android.permission.READ_CONTACTS" />`
- Para la actividad donde se van a desplegar los contactos, se define un ListView
- Los contactos en android se acceden a través de un Cursor:

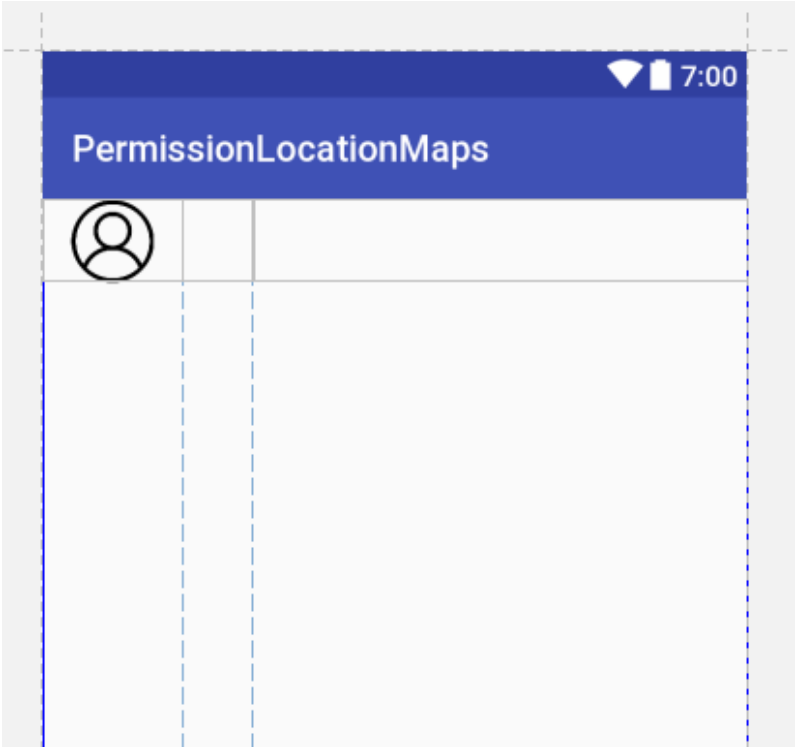
```
mProjection = new String[]{
                ContactsContract.Profile._ID,
                ContactsContract.Profile.DISPLAY_NAME_PRIMARY
        };
mCursor=getContentResolver().query(ContactsContract.Contacts.CONTENT_URI,
mProjection, null, null, null);
```

  - En el cursor está la primera posición de la tabla de contactos que tiene las columnas **id** y **display_name_primary**
  - Sólo funciona una vez el usuario haya aprobado el permiso!

Creación de un Adapter Personalizado

- Para mostrar los datos del cursor en un ListView, es necesario crear un adapter personalizado. Primero se crea un layout para cada posición de la lista en un xml separado (New -> Layout File)

```
<LinearLayout
    android:orientation="horizontal"
    …>
    <ImageView …/>
    <TextView
        android:id="@+id/idContacto"
        android:layout_width="0dp"
        android:layout_height="45dp"
        android:layout_weight="0.1"
        android:gravity="center" />
    <TextView
        android:id="@+id/nombre"
        android:layout_height="45dp"
        android:layout_width="0dp"
        android:layout_weight="0.7"
        android:gravity="center|left"/>
</LinearLayout>
```



Creación de un Adapter Personalizado

```
public class ContactsAdapter extends CursorAdapter{
    private static final int CONTACT_ID_INDEX = 0;
    private static final int DISPLAY_NAME_INDEX = 1;

    public ContactsCursor(Context context, Cursor c, int flags) {
        super(context, c, flags);
    }
    @Override
    public View newView(Context context, Cursor cursor, ViewGroup parent) {
        return LayoutInflater.from(context)
         .inflate(R.layout.item_contactos, parent, false);
    }
    @Override
    public void bindView(View view, Context context, Cursor cursor) {
        TextView tvIdContacto = (TextView) view.findViewById(R.id.idContacto);
        TextView tvNombre = (TextView) view.findViewById(R.id.nombre);
        int idnum = cursor.getInt(CONTACT_ID_INDEX);
        String nombre = cursor.getString(DISPLAY_NAME_INDEX);
        tvIdContacto.setText(String.valueOf(idnum));
        tvNombre.setText(nombre);
    }
}
```

Utilización del Adapter personalizado

```
String[] mProjection;
Cursor mCursor;
ContactsAdapter mContactsAdapter;
ListView mlista;
```

```
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_contact);
    mlista = (ListView) findViewById(R.id.list);
   mProjection = new String[]{
            ContactsContract.Profile._ID,
            ContactsContract.Profile.DISPLAY_NAME_PRIMARY,
        };
   mContactsAdapter = new ContactsAdapter(this, null, 0);
   mlista.setAdapter(mContactsAdapter);
   initView();
}
```

El cursor está en nulo hasta que el usuario acepte el permiso…

Método que hace el query si hay permiso

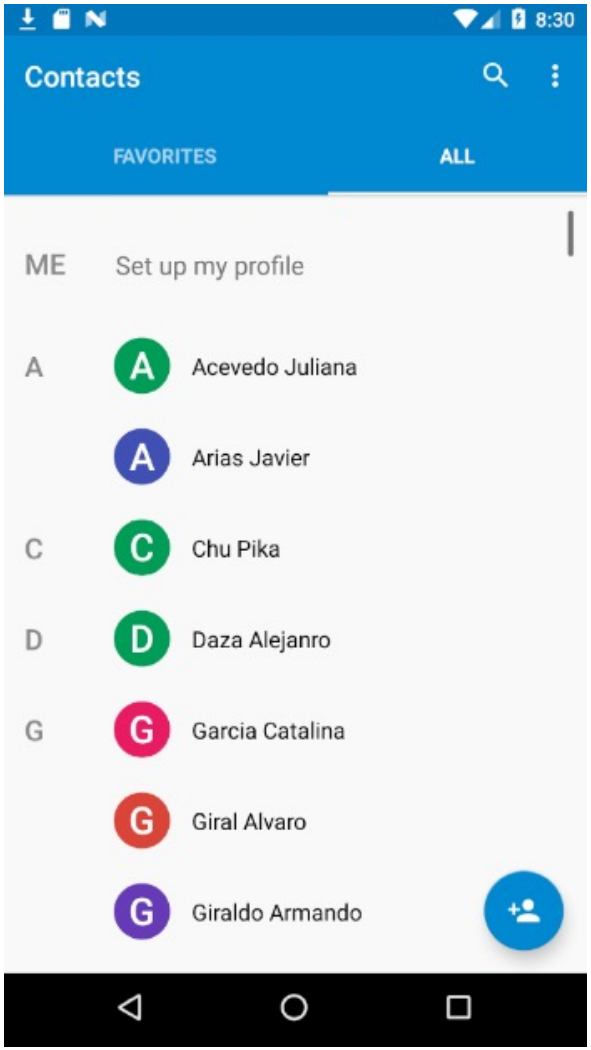
Utilización de adapter personalizado

Y cuando el usuario autorice el permiso a los contactos:

```
public void initView(){
    if(ContextCompat.checkSelfPermission(this,
Manifest.permission.READ_CONTACTS)==
PackageManager.PERMISSION_GRANTED){
                mCursor=getContentResolver().query(
                        ContactsContract.Contacts.CONTENT_URI,
                        mProjection, null, null, null);
        mContactsAdapter.changeCursor(mCursor);
    }
}
```

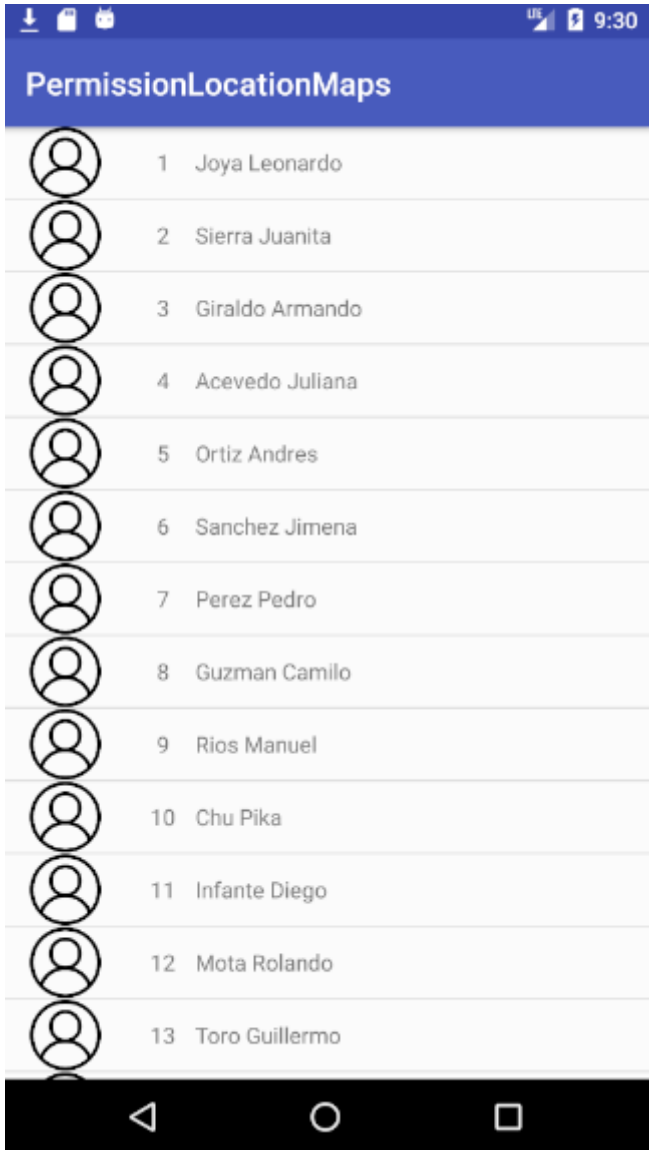
Agregar datos al emulador

- “Drag and Drop” del archivo seleccionado sobre la ventana del emulador
- Ir a Settings->Storage y remontar la tarjeta SD
- Para el archivo vcf, importar los contactos desde la aplicación de contactos



Ejercicio 2

- Importe el archivo de contactos disponible en UVirtual al emulador a través de la aplicación de contactos.
- Cambie la segunda actividad del ejercicio anterior para que sea un ListView
- Programe el botón de Contactos para que la aplicación verifique y solicite en caso de ser necesario el permiso para leer los contactos del dispositivo.
- Una vez el permiso sea aceptado, muestre los contactos del dispositivo en la lista como se ve en la figura.



Actividades con Resultado

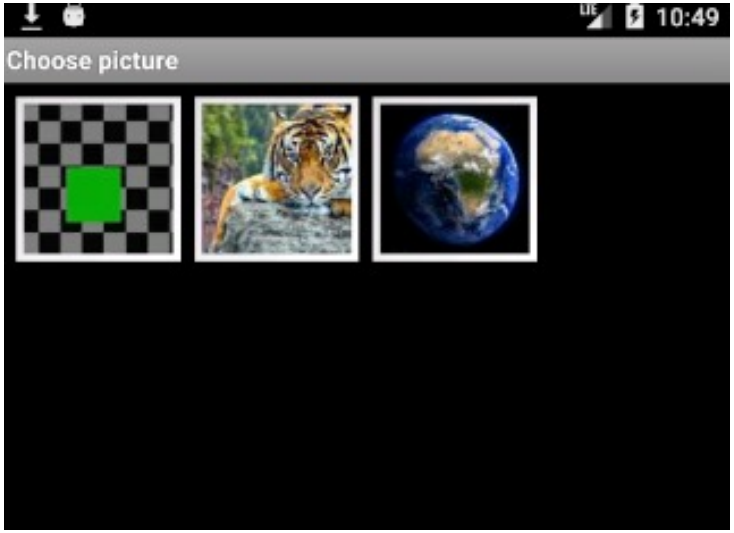
- En algunos casos es necesario recibir un resultado de una actividad que se lanza.
- En estos casos se debe usar startActivityForResult() (en lugar de startActivity()).
- Por ejemplo se puede lanzar una aplicación que tome una fotografía, y recibir esta fotografía como resultado.
- Por supuesto, la actividad lanzada debe estar programada para retornar un resultado y lo hace a través de otro objeto ***Intent.***

Abrir una aplicación de imágenes

```
Intent pickImage = new Intent(Intent.ACTION_PICK);
pickImage.setType("image/*");
startActivityForResult(pickImage, IMAGE_PICKER_REQUEST);
```

Requiere el permiso! se deben hacer las mismas verificaciones del caso de contactos

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



Galería de Android

Resultado de la Actividad

- El resultado se procesa en el callback onActivityResult

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
   super.onActivityResult(requestCode, resultCode, data);
   switch(requestCode) {
      case IMAGE_PICKER_REQUEST:
         if(resultCode == RESULT_OK){
            try {
               final Uri imageUri = data.getData();
               final InputStream imageStream = getContentResolver().openInputStream(imageUri);
               final Bitmap selectedImage = BitmapFactory.decodeStream(imageStream);
               image.setImageBitmap(selectedImage);
            } catch (FileNotFoundException e) {
               e.printStackTrace();
            }
         }
   }
}
```

La imágen resultado de la galería se carga en el objeto **image** de tipo ImageView

Resultado de la Actividad

O lanzar la cámara!

- Indicar que la aplicación requiere la cámara y evitar que la aplicación se instale en un dispositivo sin el hardware

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

- Pedir permiso!

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

https://developer.android.com/training/camera/photobasics

Intent y resultado

```
static final int CAMERA_REQUEST = 1;

private void takePicture() {
   Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
   try {
        startActivityForResult(takePictureIntent, CAMERA_REQUEST);
    } catch (ActivityNotFoundException e) {
        Log.e("PERMISSION_APP", e.getMessage());
    }
}

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

https://developer.android.com/training/camera/photobasics

Activity Result API

- ~~startActivityForResult()~~ is now deprecated!
  - Still available but recommended to use the new API
- There is a new alternative to work with results from activities and with permission requests
- Based on contracts (ActivityResultContract) that define the input type needed to produce a result along with the output type of the result.
- The APIs provide default contracts for basic intent actions like taking a picture, requesting permissions, and so on.
- You can also create your own custom contracts.

https://developer.android.com/training/basics/intents/result

Activity Resolution API
Request a permission

```
ActivityResultLauncher<String> getSinglePermission = registerForActivityResult(
        new ActivityResultContracts.RequestPermission(),
        new ActivityResultCallback<Boolean>() {
            @Override
            public void onActivityResult(Boolean result) {
                if(result == true){
                    //granted
                }else{
                    //denied
                }
            }
});
```

```
//onCreate
getSinglePermission.launch(Manifest.permission.READ_CONTACTS);
```

Activity Result API
Selecting an Image from the Gallery

- First define the contract with the callback

```
ActivityResultLauncher<String> mGetContentGallery = registerForActivityResult(
        new ActivityResultContracts.GetContent(),
        new ActivityResultCallback<Uri>() {
            @Override
            public void onActivityResult(Uri uriLocal) {
                //Load image on a view…
            }
        });
```

And invoke it whenever a user wants to select an image

```
mGetContentGallery.launch("image/*");
```

Activity Result API
Taking a Photo using the Camera

- First define the contract

```
ActivityResultLauncher<Uri> mGetContentCamera =
registerForActivityResult(new ActivityResultContracts.TakePicture(),
        new ActivityResultCallback<Boolean>() {
            @Override
            public void onActivityResult(Boolean result) {
                //Load image on a view
                setImage(uriCamera);
            }
        });
```

- *uriCamera* is an attribute defined by our application and sent to the camera, so that it can take the picture and store it there

```
Uri uriCamera;
```

Activity Result API
Taking a Photo using the Camera

- In order for the URI to work, it has to be initialized using a file provider before launching the camera:

```
File file = new File(getFilesDir(), "picFromCamera");
uriCamera = FileProvider.getUriForFile(this,
        getApplicationContext().getPackageName() + ".fileprovider", file);
mGetContentCamera.launch(uriCamera);
```

- The file provider has to be declared in the manifest inside the application tag

```
<provider
   android:name="androidx.core.content.FileProvider"
   android:authorities="com.example.testapp.fileprovider"
   android:exported="false"
   android:grantUriPermissions="true">
   <meta-data
      android:name="android.support.FILE_PROVIDER_PATHS"
      android:resource="@xml/provider_paths" />
</provider>
```

Activity Result API
Taking a Photo using the Camera

- Finally, define a new xml folder inside res, and create an xml file called provider_paths.xml

```
<paths xmlns:android="http://schemas.android.com/apk/res/android">
   <files-path
       name="files"
       path="." />
</paths>
```

Activity Resolution API

¿No permission needed?

- The reason this works is that your app is not getting access to any of:
  - The camera itself
  - The user's internal/external storage
  - The user's existing photos
- All you're getting is the opportunity to ask the user to take a photo and return that photo to you. Because the user needs to affirmatively consent as part of the process, no permission is required.
- This model also has the benefit of allowing apps to capture photos even if the user would not entrus t them with camera or storage permissions. They can rest assured that the app only got access to the single photo that the user took. The same principle applies when asking the user to select a photo as well.

https://stackoverflow.com/questions/61276633/camera-permissions-are-not-requested-but-are-still-working

Ejercicio 3

- Utilizando el Android Device Monitor, cargue algunas imágenes en el emulador de pruebas.
- Modifique la aplicación principal del ejercicio 1 para lanzar una actividad con un espacio para una imagen y dos botones de selección de imágenes y de cámara.
- Programe esta nueva actividad para verificar y solicitar los permisos de acceso a la galería de imágenes o a la cámara según sea el caso.
- Cuando se pulse el botón de selección de imágenes lance una actividad que permita buscar archivos de tipo imagen. Cuando la actividad externa retorne el resultado, utilice la imagen seleccionada y muéstrela en un Imageview en la pantalla utilizando un layout de su preferencia.
- Cuando se pulse el botón de cámara, lance la actividad para tomar una foto. Ubique la imagen que se obtiene de la cámara en el mismo ImageView que utilizó para el punto anterior.