


Manejo de Permisos en Android Studio con Java

1. Introducción: Tipos de Permisos

Android clasifica los permisos en dos categorías principales que definen cómo deben ser solicitados:

Categoría de Permiso	Descripción	Manejo en Java
Permisos de Instalación	Se otorgan automáticamente al usuario al instalar la aplicación (no requieren confirmación en <i>runtime</i>).	Solo se declaran en el <code>AndroidManifest.xml</code> .
Permisos Peligrosos	Otorgan acceso a datos sensibles (ej. contactos, cámara, ubicación). El usuario debe aprobarlos en tiempo de ejecución (<i>runtime</i>).	Deben ser declarados en el <i>Manifest</i> y solicitados al usuario con código Java/Kotlin.

 Los **Permisos Peligrosos** son obligatorios para aplicaciones que apuntan a Android 6.0 (API 23) y superiores.

2. Paso 1: Declaración en el `AndroidManifest.xml`

Antes de usar cualquier permiso, ya sea de instalación o peligroso, debes declararlo en el archivo `AndroidManifest.xml`.

XML

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

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

<application ...>
</application>
```

3. Paso 2: Implementación de Permisos Peligrosos (Runtime)

Para los permisos peligrosos, debes seguir un proceso de 3 pasos en tu código Java. Usaremos el permiso de **CÁMARA** como ejemplo.

A. Define el Código de Solicitud

Necesitarás una constante para identificar la respuesta de tu solicitud de permiso.

Java

```
public class MainActivity extends AppCompatActivity {
    private static final int CAMERA_PERMISSION_CODE = 101;

    // ...
}
```

B. Verifica y Solicita el Permiso

Este es el paso donde compruebas si el permiso ya fue concedido y, si no lo fue, lo solicitas.

Java

```

private void checkCameraPermission() {
    // 1. Verifica si el permiso ya está concedido.
    if (ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA)
        != PackageManager.PERMISSION_GRANTED) {

        // 2. Si no está concedido, solicita el permiso al usuario.
        // También puedes verificar ActivityCompat.shouldShowRequestPermissionRationale()
        // para explicar por qué necesitas el permiso.
        ActivityCompat.requestPermissions(this,
            new String[]{Manifest.permission.CAMERA},
            CAMERA_PERMISSION_CODE);
    } else {
        // 3. El permiso ya está concedido, puedes iniciar la operación.
        iniciarCamara();
    }
}

```

C. Maneja la Respuesta del Usuario

Debes sobrescribir el método onRequestPermissionsResult para saber si el usuario aceptó o denegó la solicitud.

Java

```

@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);

    if (requestCode == CAMERA_PERMISSION_CODE) {
        // Verifica si la solicitud fue para la CÁMARA y si fue concedida
        if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            // Permiso concedido

```

```

        Toast.makeText(this, "Permiso de Cámara concedido", Toast.LENGTH_SHORT).show();
        iniciarCamara();
    } else {
        // Permiso denegado
        Toast.makeText(this, "Permiso de Cámara denegado. No se puede usar la función.", Toast.LENGTH_LONG).show();
    }
}
}

private void iniciarCamara() {
    // Código para abrir la cámara...
    Log.d("Permisos", "¡Cámara iniciada!");
}

```

4. Práctica Recomendada: Explicación Racional

Si el usuario deniega un permiso y luego intenta realizar la misma acción, Android ofrece la posibilidad de mostrar una **explicación racional** antes de volver a solicitar el permiso.

Utiliza `ActivityCompat.shouldShowRequestPermissionRationale()`:

Java

```

if (ActivityCompat.shouldShowRequestPermissionRationale(this, Manifest.permission.CAMERA)) {
    // Muestra un Dialog o un Snackbar explicando por qué tu app necesita este permiso.
    new AlertDialog.Builder(this)
        .setTitle("Necesitas este permiso")
        .setMessage("La aplicación necesita acceso a la cámara para tomar fotos.")
        .setPositiveButton("OK", (dialog, which) -> {

```

```
// Vuelve a solicitar el permiso después de la explicación
ActivityCompat.requestPermissions(MainActivity.this,
    new String[]{Manifest.permission.CAMERA},
    CAMERA_PERMISSION_CODE);
})
.setNegativeButton("Cancelar", null)
.show();
} else {
    // Si ya marcó "No volver a preguntar", solicitamos directamente.
    ActivityCompat.requestPermissions(this,
        new String[]{Manifest.permission.CAMERA},
        CAMERA_PERMISSION_CODE);
}
```

5. Consejos del Experto: Android Jetpack (Permissions API)

Para un manejo de permisos más limpio y moderno, especialmente cuando se usa Kotlin o se migra código, se recomienda encarecidamente la biblioteca **AndroidX Activity/Fragment KTX** y sus utilidades para manejar los resultados de los permisos de forma reactiva (usando *Contracts* como `RequestPermission` o `RequestMultiplePermissions`).

Aunque estamos en el contexto de Java, si tu proyecto es moderno, considera estas APIs para **eliminar el código boilerplate** del `onRequestPermissionsResult`.

Ventajas de las APIs modernas:

- **Sin Sobrescritura de Método:** Evita tener que sobrescribir el método `onRequestPermissionsResult`.
- **Código Limpio:** El manejo de la respuesta está en un *callback* que se define justo donde solicitas el permiso.

Este enfoque asegura que tu aplicación sigue las directrices de seguridad de Android y ofrece una buena experiencia al usuario.