SEMINARIO DE LENGUAJES OPCIÓN ANDROID

Cámara, GPS y gestión de permisos

Esp. Fernández Sosa Juan Francisco

Acceso a la cámara de fotos

Cámara de fotos

- Se puede tener acceso a la cámara de fotos invocando alguna aplicación instalada que cumpla tal propósito.
- Es posible evitar que la app sea ofrecida para instalar en dispositivos que no dispongan de cámara de fotos.

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

Cámara de fotos

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

- Si la propiedad `android:required` vale `true` la aplicación podrá ser instalada **solo** en dispositivos que tengan cámara de fotos.
- Por el contrario, si la propiedad `android:required` vale `false` la aplicación podrá ser instalada en **cualquier** dispositivo (disponga o no de cámara de fotos).
  - El desarrollador deberá validar en tiempo de ejecución si el dispositivo cuenta con la característica, y en caso de que no, limitar el funcionamiento de la aplicación
    - ***packageManager*.hasSystemFeature(PackageManager.*FEATURE_CAMERA*)**

Actividad guiada

- Crear un nuevo proyecto Android Studio llamado "CamaraDeFotos"
- En el AndroidManifest.xml incorporar:

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

- En activity_main.xml definir la siguiente vista:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Sacar foto"
        android:onClick="sacarFoto"/>
</LinearLayout>
```

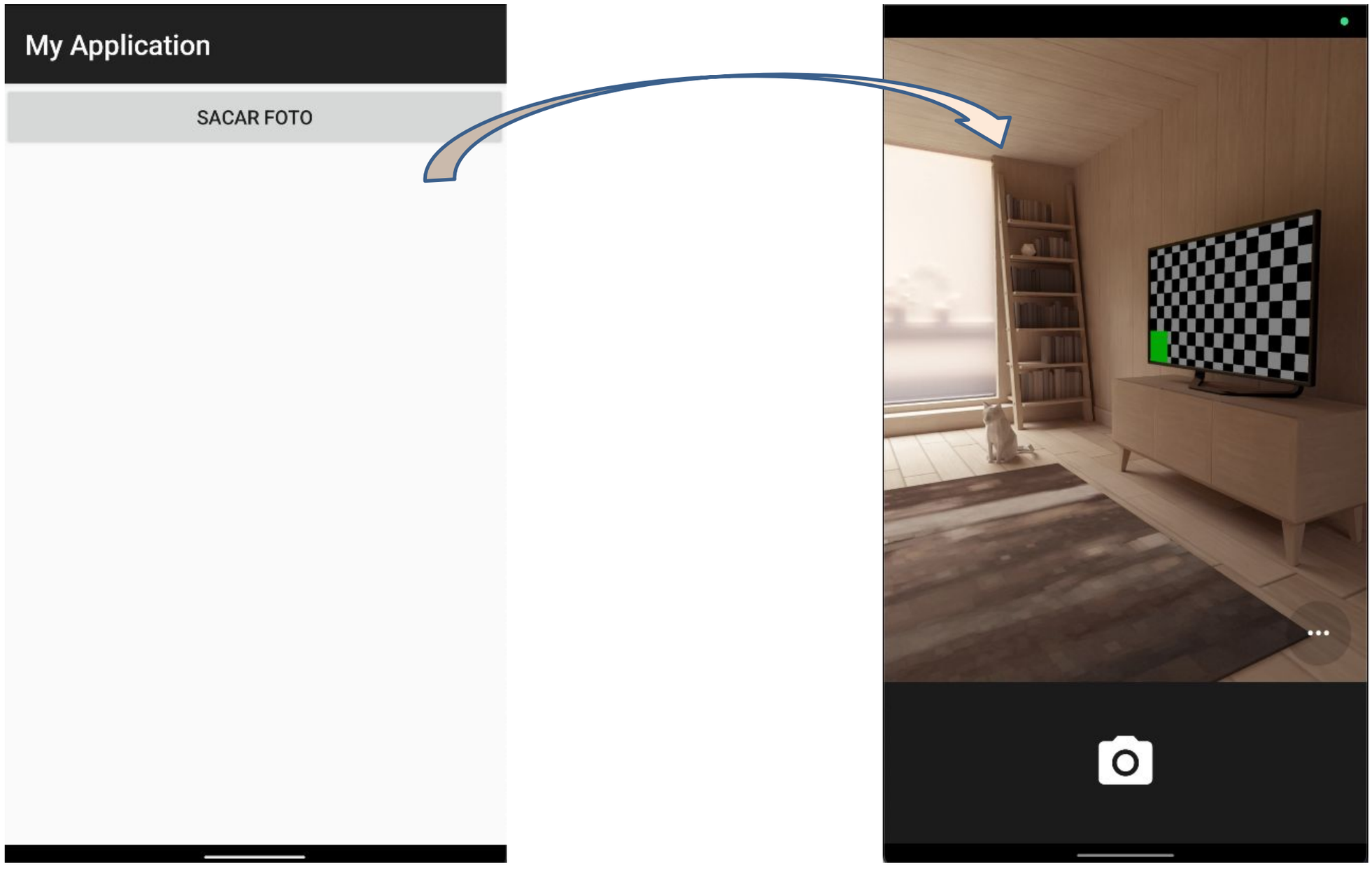
Actividad guiada

- En lugar de implementar la funcionalidad “cámara” en la app, vamos a utilizar un **Intent implícito**, para poder utilizar alguna aplicación instalada en el dispositivo que sepa responder a este comportamiento.
- La acción del intent a utilizar será **MediaStore.*ACTION_IMAGE_CAPTURE*** (esto permite que no sea necesario declarar el permiso en el Manifest)
- Incorporar el método **sacarFoto()** en la clase MainActivity

```
fun sacarFoto(v: View){
    val i = Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    startActivity(i);
}
```

Actividad guiada

- Probar en el emulador o en su dispositivo



Actividad guiada

- Se continuará mostrando un thumbnail de la imagen capturada. Agregar en el **activity_main.xml** la definición de un **ImageView** donde se mostrará el thumbnail:

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

        ...
    <ImageView
         android:layout_width="200dp"
         android:layout_height="200dp"
         android:layout_gravity="center"
         android:id="@+id/imageView1"/>
</LinearLayout>
```

Actividad guiada

- Cuando el intent termine de ejecutarse, devolverá a la actividad la imagen tomada. Hay que esperar una respuesta de ese Intent.
- Para recibir un resultado desde la activity que se va a iniciar se debe lanzar dicha activity utilizando un objeto de la clase "**ActivityResultLauncher**".
  - Registrar un escuchador de evento, que será llamado por el sistema operativo, cuando la actividad lanzada finalice

```
val cameraIntent =
registerForActivityResult(ActivityResultContracts.StartActivityForResult()
) {
        result ->
    if (result.resultCode == RESULT_OK) {
        val data = result.data;
        val imagePreview = findViewById<ImageView>(R.id.imageView1);
        val imageBitmap = data?.extras?.getParcelable<Bitmap>("data")
        imagePreview.setImageBitmap(imageBitmap)
    }
}
```

Actividad guiada

```
val cameraIntent =
registerForActivityResult(ActivityResultContracts.StartActivityForResult()
) {
    result ->
if (result.resultCode == RESULT_OK) {
    val data = result.data;
    val imagePreview = findViewById<ImageView>(R.id.imageView1);
    val imageBitmap = data?.extras?.getParcelable<Bitmap>("data")
    imagePreview.setImageBitmap(imageBitmap)
}
```

Verifico que la actividad tenga resultado OK (se tomó la fotografía)

Actividad guiada

```
val cameraIntent =
registerForActivityResult(ActivityResultContracts.StartActivityForResult()
) {
    result ->
if (result.resultCode == RESULT_OK) {
    val data = result.data;
    val imagePreview = findViewById<ImageView>(R.id.imageView1);
    val imageBitmap = data?.extras?.getParcelable<Bitmap>("data")
    imagePreview.setImageBitmap(imageBitmap)
}
```

Se obtiene la referencia al ImageView para luego asociar la imagen nueva

Actividad guiada

```
val cameraIntent =
registerForActivityResult(ActivityResultContracts.StartActivityForResult()
) {
    result ->
if (result.resultCode == RESULT_OK) {
    val data = result.data;
    val imagePreview = findViewById<ImageView>(R.id.imageView1);
    val imageBitmap = data?.extras?.getParcelable<Bitmap>("data")
    imagePreview.setImageBitmap(imageBitmap)
}
```

Se obtiene el Bitmap (mapa de bits) asociado a la imagen

Actividad guiada

```
val cameraIntent =
registerForActivityResult(ActivityResultContracts.StartActivityForResult()
) {
        result ->
    if (result.resultCode == RESULT_OK) {
        val data = result.data;
        val imagePreview = findViewById<ImageView>(R.id.imageView1);
        val imageBitmap = data?.extras?.getParcelable<Bitmap>("data")
        imagePreview.setImageBitmap(imageBitmap)
    }
}
```

Se establece el bitmap de la imagen a mostrar en el ImageView, utilizando el método **setImageBitmap()**

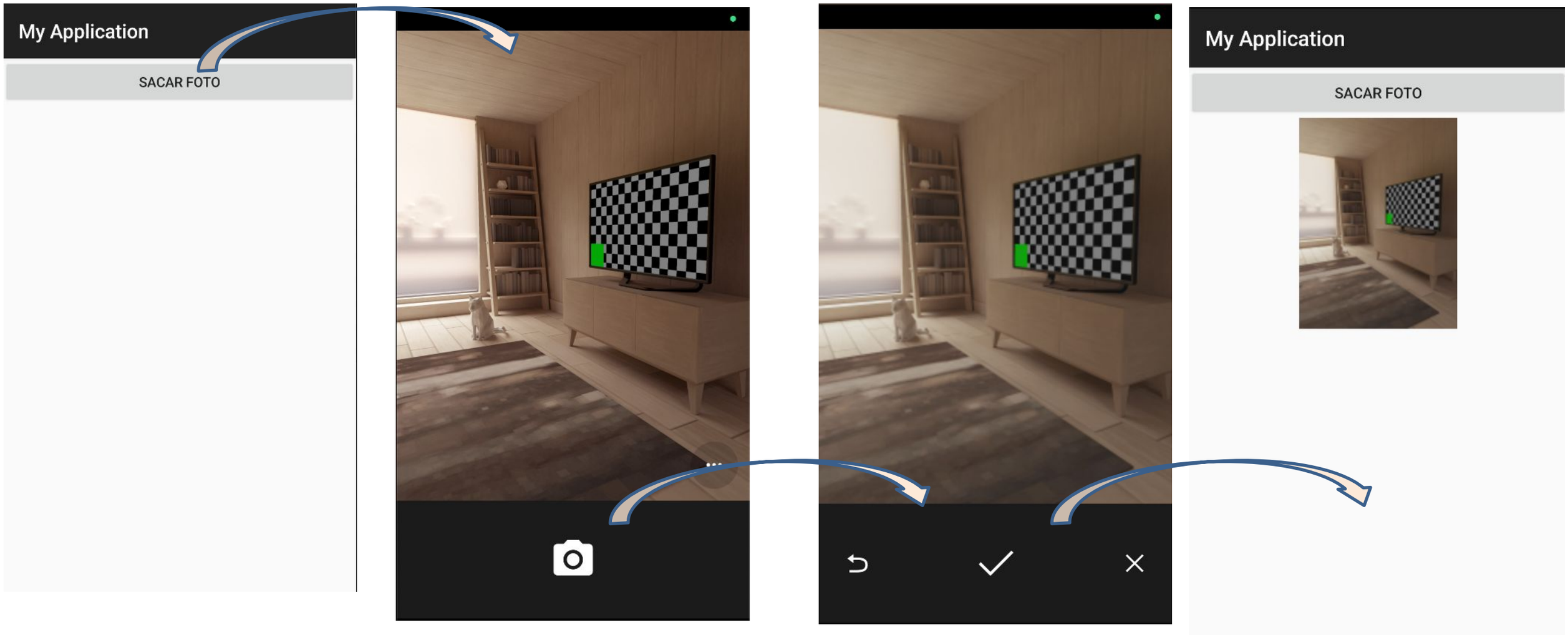
Actividad guiada

- lanzar el intent implícito utilizando el método "**launch()**" del objeto **ActivityResultLauncher**

```
fun sacarFoto(v: View){
    val i = Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    cameraIntent.launch(i);
}
```

Actividad guiada

- Probar en el emulador o en su dispositivo



Localización Geográfica

Localización geográfica

- Para conocer la **ubicación** del dispositivo hay que **solicitar permisos** al usuario ya que es un **dato sensible**.
- **Los permisos** que requerirá la app se deben **indicar** en el archivo **Manifest**.
- En el caso particular de la **ubicación** hay **dos tipos de permisos** que se pueden incluir en el manifest
  - **ACCESS_COARSE_LOCATION**: Este permiso permite a una aplicación acceder a la ubicación aproximada del dispositivo. La precisión de la ubicación puede ser de nivel de ciudad o de nivel de área amplia.
  - **ACCESS_FINE_LOCATION**: Este permiso permite a una aplicación acceder a la ubicación exacta y precisa del dispositivo.

Localización geográfica

- Además de incluirlos en el Manifest, **los permisos se gestionan en tiempo de ejecución**. Hay que **verificar desde el código** de la app que el usuario haya dado el permiso y en caso que no sea así, se podrá solicitar que lo haga.
- Para saber si se tiene o no un determinado permiso se utiliza el método **ContextCompat.checkSelfPermission()**
- Para solicitar un permiso en particular se utiliza el método **ActivityCompat.requestPermission()**

Google Play Services

- Para obtener la ubicación actual del dispositivo en el que se está ejecutando una aplicación, se debe disponer de Google Play Services
  - En Android Studio, en el menú Tools hacer click en SDK Manager
    - hacer click en SDK Tools
    - seleccionar Google Play Services
    - hacer click en OK.
- Se podrá probar en
  - Dispositivos que tengan Android 4.0 o superior, y que incluyan Google Play Store
  - Emuladores con Android 4.2.2 o superior con Google APIs

Localización geográfica - Actividad Guiada

- Crear un nuevo proyecto desde Android Studio
- Se deben indicar los permisos a través del archivo Manifest.

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

- En build.gradle (nivel de aplicación) se debe establecer la dependencia con la librería de localización de Google Play Services.

```
dependencies {
     ….
      implementation("com.google.android.gms:play-services-location:21.2.0")
     ….
}
```

Localización geográfica - Actividad Guiada

- Copiar el siguiente código en la clase MainActivity

```
private fun checkLocationPermission(): Boolean {
    val coarseLocation =
ContextCompat.checkSelfPermission(this,
android.Manifest.permission.ACCESS_COARSE_LOCATION)
    val fineLocation = ContextCompat.checkSelfPermission(this,
android.Manifest.permission.ACCESS_FINE_LOCATION)

    if (coarseLocation != PackageManager.PERMISSION_GRANTED ||
fineLocation != PackageManager.PERMISSION_GRANTED) {
        requestLocationPermission()
        return false
    }
    return true
}
```

Localización geográfica - Actividad Guiada

- Cop

Se obtiene el valor que tiene la app de los permisos **ACCESS_COARSE_LOCATION** y **ACCES_FINE_LOCATION**

```
private fun checkLocationPermission(): Boolean {
    val coarseLocation =
ContextCompat.checkSelfPermission(this,
android.Manifest.permission.ACCESS_COARSE_LOCATION)
    val fineLocation = ContextCompat.checkSelfPermission(this,
android.Manifest.permission.ACCESS_FINE_LOCATION)

    if (coarseLocation != PackageManager.PERMISSION_GRANTED ||
fineLocation != PackageManager.PERMISSION_GRANTED) {
        requestLocationPermission()
    }
    return true
}
```

Verifica el estado de los permisos para el usuario

Localización geográfica - Actividad Guiada

- Copiar el siguiente código en la clase MainActivity

```
private fun requestLocationPermission() {
    ActivityCompat.requestPermissions(this,
arrayOf(android.Manifest.permission.ACCESS_COARSE_LOCATION,
android.Manifest.permission.ACCESS_FINE_LOCATION), 321)
}
```

Localización geográfica - Actividad Guiada

- Copiar el siguiente código en la clase MainActivity

El método requestPermissions() muestra un diálogo en la pantalla para que el usuario gestione los permisos

```
private fun requestLocationPermission() {
    ActivityCompat.requestPermissions(this,
arrayOf(android.Manifest.permission.ACCESS_COARSE_LOCATION,
android.Manifest.permission.ACCESS_FINE_LOCATION), 321)
}
```

Código que permite identificar la solicitud de permiso. Se lo puede declarar como constante en la clase MainActivity, lo utilizaremos para conocer el resultado de la solicitud

Localización geográfica - Actividad Guiada

```
override fun onRequestPermissionsResult(
    requestCode: Int,
    permissions: Array<out String>,
    grantResults: IntArray)
{
    super.onRequestPermissionsResult(requestCode, permissions,
grantResults)
    if (requestCode == PERMISSION_REQUEST_CODE) {
        if (grantResults.isNotEmpty() && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
            // Permiso concedido
            obtenerUbicacion(null);
        } else {
            // Permiso denegado
            // Realizar acciones adicionales o mostrar un mensaje de
error
        }
    }
}
```

Agregar este método en la clase MainActivity

Localización geográfica - Actividad Guiada

```
override fun onRequestPermissionsResult(
    requestCode: Int,
    permissions: Array<out String>,
    gran
{
    supe
grantResults)
    if (requestCode == PERMISSION_REQUEST_CODE) {
        if (grantResults.isNotEmpty() && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
            // Permiso concedido
        } else {
            // Permiso denegado
            // Realizar acciones adicionales
error
        }
    }
}
```

En la aplicación se podrían necesitar diferentes permisos, con el requestCode podemos identificarlos

Se otorgó el permiso

No se concedió el permiso solicitado

Agregar este método en la clase MainActivity

Localización geográfica - Actividad Guiada

- Editar el archivo activity_main.xml

```
<LinearLayout
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    android:orientation="vertical"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <TextView
        android:id="@+id/latitudTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Consultando latitud..." />
    <TextView
        android:id="@+id/longitudTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Consultando longitud..." />

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Obtener ubicación"
        android:onClick="obtenerUbicacion"/>

</LinearLayout>
```

Localización geográfica - Actividad Guiada

- Agregar una función a la clase **MainActivity** que reciba como parámetro un tipo de dato **Location** y actualice los valores de los TextViews.
- Este método será llamado cada vez que se tenga una actualización de la ubicación del dispositivo

```
fun imprimirUbicacion(location: Location){
    var latitudeTV = findViewById<TextView>(R.id.latitudTextView);
    var longitudeTV = findViewById<TextView>(R.id.longitudTextView);
    latitudeTV.setText(location.latitude.toString());
    longitudeTV.setText(location.longitude.toString());
}
```

Localización geográfica - Actividad Guiada

- Para conocer la ubicación del dispositivo vamos a utilizar tres clases:
  - **FusedLocationProviderClient**: permite conocer la última ubicación del dispositivo (lastLocation) y también la ubicación en tiempo real (requestLocationUpdates).
  - **LocationRequest**: permite construir una solicitud de ubicación, indicando con qué intervalo, en milisegundos, se desea obtener dicho dato.
  - **LocationCallback**: Es el receptor de las ubicaciones en tiempo real.
    - Su método onLocationResult(...) se ejecuta cada vez que hay una nueva posición.

Localización geográfica - Actividad Guiada

```
fun obtenerUbicacion(v: View?) {
    if (!checkLocationPermission()) return
    val fusedLocationClient = LocationServices.getFusedLocationProviderClient(this)
    if (ActivityCompat.checkSelfPermission(
            this,
            android.Manifest.permission.ACCESS_FINE_LOCATION
        ) != PackageManager.PERMISSION_GRANTED
    ) {
        Toast.makeText(this, "Permiso de ubicación no concedido",
Toast.LENGTH_SHORT).show()
        return
    }
```

Se obtiene una instancia de la clase **FusedLocationProviderClient**

Verificación de permisos

```
    val locationRequest = LocationRequest.Builder(
        Priority.PRIORITY_HIGH_ACCURACY,
        3000 // cada 3 segundos
    ).build()

    val locationCallback = object : LocationCallback() {
        override fun onLocationResult(result: LocationResult) {
            val location = result.lastLocation
            if (location != null) {
                imprimirUbicacion(location)
            } else {
                Toast.makeText(this@MainActivity, "Ubicación no disponible",
Toast.LENGTH_SHORT).show()
            }
        }
    }
    fusedLocationClient.requestLocationUpdates(
        locationRequest,
        locationCallback,
        Looper.getMainLooper()
    )
}
```

Se crea una instancia de **LocationRequest** con un intervalo de 3 segundos

Cada 3 segundos se va a ejecutar el Callback **LocationCallback**()

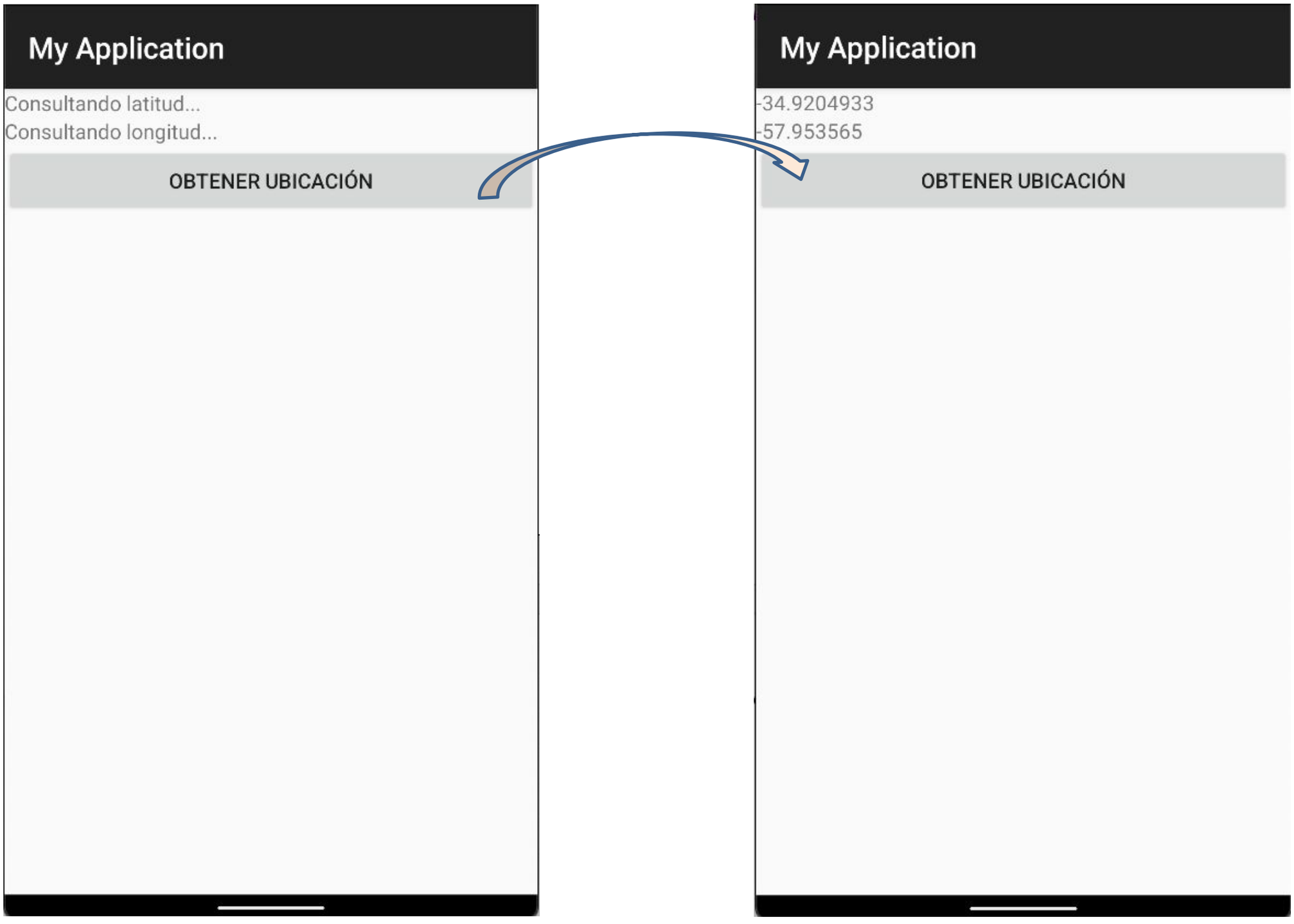
Se solicita a la clase **FusedLocationProviderClient** tener actualizaciones de la ubicación

Localización geográfica - Actividad Guiada

```
fun obtenerUbicacion(v: View?) {
    if (!checkLocationPermission()) return
    val fusedLocationClient = LocationServices.getFusedLocationProviderClient(this)

    // Verificamos que el permiso haya sido otorgado
    if (ActivityCompat.checkSelfPermission(
            this,
            android.Manifest.permission.ACCESS_FINE_LOCATION
        ) != PackageManager.PERMISSION_GRANTED
    ) {
        Toast.makeText(this, "Permiso de ubicación no concedido", Toast.LENGTH_SHORT).show()
        return
    }

    // Configuramos la solicitud de ubicación (intervalo en milisegundos)
    val locationRequest = LocationRequest.Builder(
        Priority.PRIORITY_HIGH_ACCURACY,
        3000 // cada 3 segundos
    ).build()

    // Creamos el callback que recibirá las actualizaciones
    val locationCallback = object : LocationCallback() {
        override fun onLocationResult(result: LocationResult) {
            val location = result.lastLocation
            if (location != null) {
                imprimirUbicacion(location)
            } else {
                Toast.makeText(this@MainActivity, "Ubicación no disponible",
Toast.LENGTH_SHORT).show()
            }
        }
    }

    // Iniciamos la solicitud de actualizaciones
    fusedLocationClient.requestLocationUpdates(
        locationRequest,
        locationCallback,
        Looper.getMainLooper()
    )
}
```

Copiar el código de la función **obtenerUbicacion()** en la clase **MainActivity**. Ejecutar y probar en el emulador

Localización geográfica - Actividad Guiada



Localización geográfica - Actividad Guiada

Cambiar la ubicación desde las opciones del emulador