Programación Multimedia y Dispositivos Móviles

UD 5. Intents y permisos

Javier Carrasco

Curso 2021 / 2022

Última actualización: septiembre de 2021.
Versión impresa en Amazon: https://www.amazon.es/dp/B08NM4XV4Q

Intents y permisos

5. Intents y permisos

Los ***Intents*** son objetos que permitirán solicitar acciones a otros componentes de la aplicación (una *activity*, un servicio, un proveedor de contenido, etc), pero también permiten iniciar actividades en otras aplicaciones, como hacer una foto o ver un mapa.

Existen *Intents* de dos tipos:

- **Explícitos**, indicarán que deben lanzar exactamente, su uso típico es ejecutar diferentes componentes internos de una aplicación. Por ejemplo, una actividad.
- **Implícitos**, se utilizan para lanzar tareas abstractas, del tipo *"quiero hacer una llamada"* o *"quiero hacer una foto"*. Estas peticiones se resuelven en tiempo de ejecución, por lo que el sistema buscará los componentes registrados para la tarea pedida, si encontrase varias, el sistema preguntará al usuario que componente prefiere.

La potencia de los *Intents* es muy grande, pero de momento se centrará la atención en los tres usos principales que se deben dominar.

- **Para abrir una actividad:**

Como ya sabes, una *activity* es una única pantalla de la aplicación. Se puede iniciar una nueva *activity*, creando una nueva instancia pasando un *Intent* mediante `startActivity()`. La *Intent* describirá que actividad se debe iniciar y los datos que hacen falta.

Si se necesita obtener un resultado al finalizar la actividad, se utilizará el método `startActivityForResult()`. La actividad que lanza la llamada, recibirá cuando finalice un objeto *Intent* separado en el *callback* de `onActivityResult()`.

- **Para iniciar un servicio:**

Un *Service* es un componente que realiza operaciones en segundo plano sin una interfaz de usuario. Se pueden iniciar un servicio para realizar una única operación, como descargar un archivo, pasando una *Intent* a `startService()`. La *Intent* describe el servicio que se debe iniciar y contendrá los datos necesarios para la tarea.

Si el servicio está diseñado con una interfaz cliente-servidor, puedes establecer un enlace con el servicio de otro componente pasando una *Intent* a `bindService()`.

- **Para entregar un mensaje:**

Se utiliza para crear mensajes de aviso que cualquier aplicación puede recibir. También el sistema operativo puede enviar mensajes de eventos producidos en el sistema, como cuando el sistema arranca o el dispositivo comienza a cargarse. Se puede enviar un mensaje a otras *apps* pasando una *Intent* a `sendBroadcast()`, `sendOrderedBroadcast()` o `sendStickyBroadcast()`. Este tipo puede ser útil para la comunicación entre *apps*.

A continuación, se verá lo más básico de los *Intents*, implícitos y explícitos, para comenzar así a dominar las acciones básicas.

5.1. Filtros de *intent*

Como ya se ha comentado, cuando se lanza un *intent* implícito, es el propio sistema operativo el que busca la manera de poder realizarlo. Para que el sistema pueda encontrar la forma de hacerlo, deberá declararse en el fichero *manifest* esta posibilidad mediante el uso de la etiqueta `<intent-filter>`.

Si observas el fichero `AndroidManifest.xml` de un proyecto, podrás ver como se le indica al sistema operativo qué actividad es la que debe lanzar a la hora de ejecutar la aplicación. Esto es gracias a `android.intent.action.MAIN`, que sería la acción aceptada y a `android.intent.category.LAUNCHER`, que sería la categoría. Si la aplicación tiene más de una *activity*, esta categoría únicamente podrá estar en una de ellas.

```
<application
    ...
    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
```

Las acciones más utilizadas tienes definida su propia constante `ACTION` dentro de la clase `Intent`. Puedes consultar las constantes en el enlace al pie[1]. Una de las más comunes es `ACTION_VIEW`, que en el *manifest* se utilizaría como `android.intent.action.VIEW`. Esta acción se utiliza para poder hacer uso de los datos que se intercambian.

Las categorías[2] también disponen de su propia constante, en esta caso `CATEGORY`, dentro de la clase `Intent`.

En el siguiente código de ejemplo, se muestra la configuración de una actividad que puede recibir datos cuando estos sean de tipo texto mediante el uso de `ACTION_SEND`.

```
<activity android:name="ShareActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

Puedes encontrar más información sobre los filtros de *intent* en la documentación oficial[3] de Android.

---

1 *Standard Activity Actions* (https://developer.android.com/reference/kotlin/android/content/Intent#standard-activity-actions)
2 *Standard Categories* (https://developer.android.com/reference/kotlin/android/content/Intent#standard-categories)
3 *Intents* y filtros de *intents* (https://developer.android.com/guide/components/intents-filters?hl=es)

5.2. Abrir una actividad nueva (*intent* explícito)

Comienza por añadir un *EditText* y un *Button* a la `activity_main.xml`, quedando como puedes ver a continuación.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <EditText
        android:id="@+id/textToSend"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="32dp"
        android:layout_marginEnd="8dp"
        android:ems="10"
        android:hint="@string/hintText"
        android:importantForAutofill="no"
        android:inputType="text"
        app:layout_constraintEnd_toStartOf="@+id/bt_send"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:targetApi="o" />

    <Button
        android:id="@+id/bt_send"
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_marginTop="32dp"
        android:layout_marginEnd="8dp"
        android:text="@string/myButton"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toEndOf="@+id/textToSend"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.0" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Fíjate en la propiedad `android:layout_marginStart`, que puede ser sustituida, o más bien complementada, por `android:layout_marignLeft` para añadir compatibilidad a la aplicación con APIs inferiores a la 17.

A continuación, se creará una segunda actividad vacía, la cual se abrirá al pulsar el botón creado en la actividad principal. Utiliza la opción ***File > New > Activity > Empty Activity***.

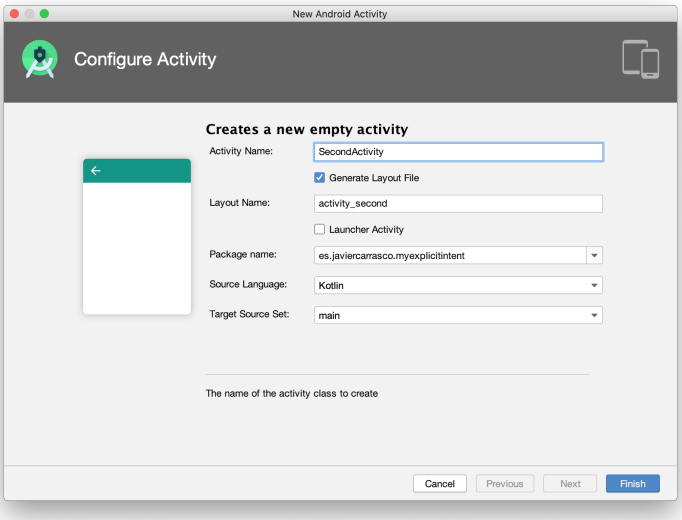


Figura 1

Esta acción añadirá automáticamente la siguiente información al fichero *Android Manifest* del proyecto.

```
<activity android:name=".SecondActivity"></activity>
```

Esta línea indica la existencia de una nueva *activity* en el proyecto. A continuación se añadirá navegación, estableciendo la jerarquía de las *activities* utilizando la propiedad `android:parentActivityName`, esto le permitirá a la aplicación saber donde volver cuando se pulse el botón *"atrás"*. Esta propiedad será la encargada de mostrar la flecha para volver en la barra de título de la *app*.

```
<activity android:name=".SecondActivity"
    android:label="Second Activity"
    android:parentActivityName=".MainActivity">
</activity>
```

La propiedad `android:label` permite cambiar el título de la *activity* mostrado en la barra de título de la aplicación. De vuelta al *layout* de la segunda actividad, `activity_second.xml`, ésta únicamente contendrá un *TextView* para mostrar el mensaje enviado.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
    tools:context=".SecondActivity">
    <TextView
        android:id="@+id/tv_msgReceived"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginLeft="16dp"
        android:layout_marginTop="32dp"
        android:layout_marginEnd="16dp"
        android:layout_marginRight="16dp"
        android:textSize="24sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:text="Texto de muestra" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

La propiedad `tools:text` no requiere crear un valor en `string.xml`, esta se utiliza para ayudar en el diseño y ver como puede quedar, no aparecerá cuando se ejecute la aplicación. Ahora se modificará la lógica de la actividad que realiza la llamada, se comenzará con lo básico, llamar a otra *activity* sin pasarle valores.

```
class MainActivity : AppCompatActivity() {
    // View Binding
    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        binding.btSend.setOnClickListener {
            // Se crea un objeto de tipo Intent.
            val myIntent = Intent(this, SecondActivity::class.java)

            // Se lanza la activity.
            startActivity(myIntent)
        }
    }
}
```

Se crea un objeto de tipo `Intent` y se le pasa el contexto y la clase `SecondActivity.kt`, aunque ésta debe especificarse como de tipo Java. El contenidos de la clase en cuestión será.

```
class SecondActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
```

```
        setContentView(R.layout.activity_second)
    }
}
```

Paso de información entre actividades

A continuación, se verá el paso de información entre actividades, lo primero que se deberá saber es que se utilizará el sistema de **clave-valor**. Se modificará la llamada haciendo uso del método `putExtra()` que usa el sistema clave-valor.

```
class MainActivity : AppCompatActivity() {
    // View Binding
    private lateinit var binding: ActivityMainBinding

    companion object {
        const val EXTRA_MESSAGE = "myMessage"
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        binding.btSend.setOnClickListener {
            // Se crea un objeto de tipo Intent.
            val myIntent = Intent(this, SecondActivity::class.java).apply {
                // Se añade la información a pasar por clave-valor.
                putExtra(EXTRA_MESSAGE, binding.textToSend.text.toString())
            }
            // Se lanza la activity.
            startActivity(myIntent)
        }
    }
}
```

Se añade una constante con la clave para el valor que vaya a ser pasado, así se evitarán posibles errores a la hora de recoger los datos. Fíjate que para crear esta variable se debe utilizar un `companion object`[4].

A continuación se muestra el código necesario para recoger los datos en la segunda actividad.

```
class SecondActivity : AppCompatActivity() {
    private lateinit var binding: ActivitySecondBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

```

4 Companion Objects (https://kotlinlang.org/docs/tutorials/kotlin-for-py/objects-and-companion-objects.html#companion-objects)

```
        binding = ActivitySecondBinding.inflate(layoutInflater)
        setContentView(binding.root)

        // Se recuperan los datos y se asignan al TextView.
        val message = intent.getStringExtra(EXTRA_MESSAGE)
        binding.tvMsgReceived.text = message
    }
}
```

Debes tener en cuenta que, el objeto `intent`, hace referencia al *intent* de la actividad que hace la llamada a la actividad que se quiere abrir. El resultado puede ser como el que se muestra en la siguiente figura.

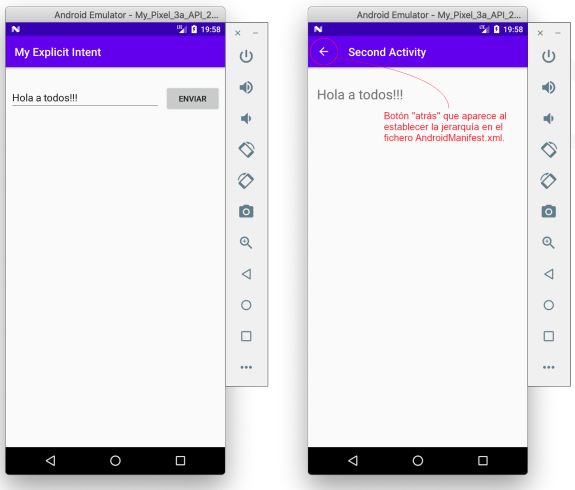


Figura 2

Control de campos vacíos

Ahora, se añadirá un mejora para evitar que se pueda abrir la segunda actividad si no se ha escrito nada en el *EditText*. Crea en la clase principal la siguiente función.

```
private fun validarTexto(): Boolean {
    var esValido = true

   if (TextUtils.isEmpty(binding.textToSend.text.toString())) {
        // Si la propiedad error tiene valor, se muestra el aviso.
        binding.textToSend.error = "Información requerida"
        esValido = false
    } else binding.textToSend.error = null

    return esValido
}
```

Los *EditText* en Android tienen una propiedad llamada `error`, a la cual, si se le asigna un texto muestra el aviso cuando se pulse sobre la exclamación de aviso.

A continuación, deberás añadir una comprobación en el *listener* antes de lanzar el *intent*, el resultado puede ser algo parecido a lo que muestra la siguiente imagen.

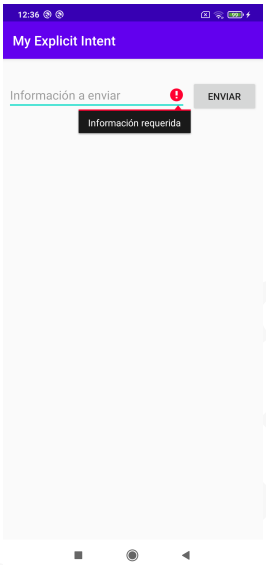


Figura 3

Devolución de un estado

Se ha visto como hacer un intercambio de información entre *activities* en una única dirección. Ahora se verá como hacer una comunicación bidireccional, ya que en ocasiones puede ser interesante recibir una respuesta de otra *activity*. Para el siguiente ejemplo se utilizará la llamada al método `startActivityForResult()`.

Para este ejemplo se creará un nuevo proyecto con dos actividades, la principal, será como la vista en el ejemplo anterior, a la cual se le añadirá un *TextView* para recibir aquello que devuelva la segunda actividad.

```
<TextView
    android:id="@+id/tv_result"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginTop="16dp"
    android:layout_marginEnd="8dp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textToSend"
    tools:text="Muestra el resultado" />
```

Figura 4

La segunda actividad (*activity_second.xml*) simulará una aceptación de condiciones, la cual permitirá devolver si se aceptan o no.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SecondActivity">

    <TextView
        android:id="@+id/tv_nameReceived"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="32dp"
        android:layout_marginEnd="16dp"
        android:textSize="18sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:text="Texto de muestra" />

    <Button
        android:id="@+id/bt_accept"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="32dp"
        android:layout_marginEnd="16dp"
        android:text="@string/btn_accept"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/tv_nameReceived" />

   <Button
        android:id="@+id/bt_cancel"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
```

```
        android:layout_marginStart="16dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="16dp"
        android:text="@string/btn_cancel"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/bt_accept" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

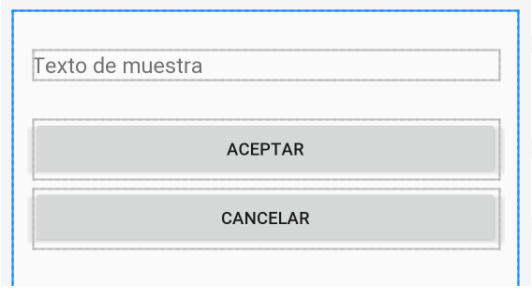


Figura 5

A continuación se verá el código **Kotlin** necesario para el intercambio de información entre dos *activities*. En este primer caso, la actividad principal recibirá de la segunda actividad si la operación es correcta o no, no se obtendrá ningún dato más.

En primer lugar, en la clase `MainActivity.kt` se deberá añadir la funcionalidad para el botón *Enviar*.

```
class MainActivity : AppCompatActivity() {
    // View Binding
    private lateinit var binding: ActivityMainBinding
    private var REQUEST_CODE = 1234

    companion object {
        const val TAG_APP = "myExplicitIntent2"
        const val EXTRA_NAME = "userNAME"
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        // Se oculta el TextView que mostrará el resultado.
        binding.tvResult.visibility = View.INVISIBLE

        binding.btSend.setOnClickListener { askConditions() }
    }

    private fun askConditions() {
```

```
        Log.d(TAG_APP, "askConditions()")

        // Se vuelve a ocultar el TV que mostrará el resultado.
        binding.tvResult.visibility = View.INVISIBLE
        // Se crea un objeto de tipo Intent.
        val myIntent = Intent(this, SecondActivity::class.java).apply {
            // Se añade la información a pasar por clave-valor.
            putExtra(EXTRA_NAME, binding.textToSend.text.toString())
        }
        // Se lanza la activity.
        startActivityForResult(myIntent, REQUEST_CODE)
    }
}
```

En este caso se utiliza la instrucción `startActivityForResult(myIntent, 1234)`, esta lanza una nueva *activity* y queda a la espera de recibir respuesta. Los parámetros son el *Intent* configurado con la *activity* que se quiere abrir, y un código de tipo entero que se asignará para identificar la respuesta cuando se produzca. No utilices siempre el mismo código para llamar a todas tus *activities*. A continuación se verá el código de `SecondActivity.kt`.

```
class SecondActivity : AppCompatActivity() {
    // View Binding
    private lateinit var binding: ActivitySecondBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivitySecondBinding.inflate(layoutInflater)
        setContentView(binding.root)

        // Se recuperan los datos y se asignan al TextView.
        val nameReceived = intent.getStringExtra(EXTRA_NAME)

        binding.tvNameReceived.text = getString(
            R.string.msgAccept,
            nameReceived
        )

        binding.btAccept.setOnClickListener {
            setResult(Activity.RESULT_OK)
            Log.d(TAG_APP, "Valor devuelto OK")
            finish()
        }

        binding.btCancel.setOnClickListener {
            setResult(Activity.RESULT_CANCELED)
            Log.d(TAG_APP, "Valor devuelto CANCELED")
            finish()
        }
    }
}
```

Para devolver el valor, aceptado o cancelado, verdadero o falso, se utiliza el método `setResult(valor)`, donde *valor* será `Activity.RESULT_OK` o `Activity.RESULT_OK`. Además, deberá finalizarse la *activity* con el método `finish()` para volver a la actividad que realizó la llamada.

Observa la línea `binding.tvNameReceived.text = getString(R.string.msgAccept, nameReceived)`, concatena el valor definido en `strings.xml` con la variable `nameReceived`. Para hacer esto debes definir la variable `msgAccept` en `strings.xml` como se muestra a continuación.

```
<string name="msgAccept">Hola %s, ¿aceptas las condiciones?</string>
```

Ahora, vuelve a `MainActivity.kt`, deberás sobrescribir el método `onActivityResult()` para poder recibir la respuesta de la actividad llamada. Puedes añadirlo utilizando la opción **Ctrl+O** o en el menú **Code > Override Methods**. También puedes utilizar la opción de auto-completar según vas escribiendo.

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)

    if (requestCode == REQUEST_CODE) {
        if (resultCode == Activity.RESULT_OK) {
            binding.tvResult.text = "Condiciones aceptadas."
            binding.tvResult.visibility = View.VISIBLE
        }

        if (resultCode == Activity.RESULT_CANCELED) {
            binding.tvResult.text = "Se canceló el contrato!"
            binding.tvResult.visibility = View.VISIBLE
        }
    }
}
```

Al sobrecargar este método, se utiliza el parámetro `requestCode` que indicará el código de la *activity* que ha finalizado y, así saber cual es y poder actuar en consecuencia. El parámetro `resultCode` indicará el resultado devuelto por la *activity* finalizada.

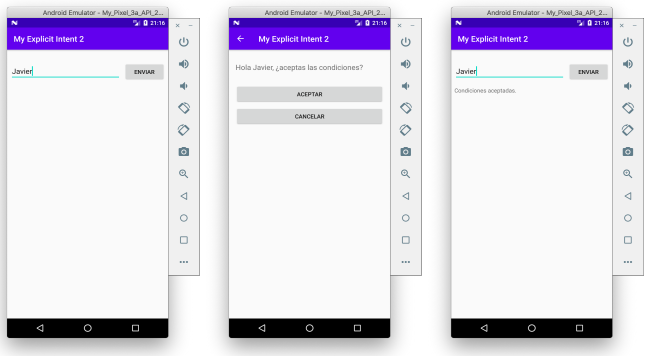


Figura 6

Intercambio de información por contrato

Si has seguido los pasos del ejemplo, te habrás dado cuenta que `startActivityForResult()` y `onActivityResult()` están *deprecated*. Esto se debe a que a partir de **AndroidX**, se cambia de estrategia para el intercambio de información entre actividades, y se utiliza para ello un sistema de contratos[5]. Este cambio se aplica para evitar el consumo excesivo de memoria, evitando así que el proceso proceso y la actividad que ha lanzado el *intent*. Observa como quedaría el ejemplo visto con este nuevo sistema, solo afectará a la creación del *intent*, no a la devolución de datos.

Todos los cambios se verán reflejados en la clase principal. Desaparece la variable `REQUEST_CODE`, ya no es necesaria, así como el método `onActivityResult()`. En su lugar, se creará el siguiente objeto que se encargará de las acciones a realizar tras recibir la respuesta.

```
// Objeto para recoger la respuesta de la activity.
var resultadoActivity = registerForActivityResult(StartActivityForResult())
  { result →

     if (result.resultCode == Activity.RESULT_OK) {
         // No se usan los REQUEST_CODE.
         val data: Intent? = result.data
         binding.tvResult.text = "Condiciones aceptadas."
         binding.tvResult.visibility = View.VISIBLE
     }

     if (result.resultCode == Activity.RESULT_CANCELED) {
         binding.tvResult.text = "Se canceló el contrato!"
         binding.tvResult.visibility = View.VISIBLE
     }
}
```

El método `askConditions()` cambia ligeramente, el mayor cambio se aprecia en el objeto `resultadoActivity` creado.

```
private fun askConditions() {
    Log.d(TAG_APP, "askConditions()")

    // Se vuelve a ocultar el TV que mostrará el resultado.
    binding.tvResult.visibility = View.INVISIBLE

   // Se crea un objeto de tipo Intent
    val myIntent = Intent(this, SecondActivity::class.java).apply {
        // Se añade la información a pasar por clave-valor.
        putExtra(EXTRA_NAME, binding.textToSend.text.toString())
    }

    resultadoActivity.launch(myIntent)
}
```

5 Cómo obtener un resultado de una actividad (https://developer.android.com/training/basics/intents/result)

Observa que ahora se lanzará el *intent* utilizando el método `launch()` del objeto creado, al cual se le deberá pasar el mismo *intent* que el visto anteriormente. El resultado de la aplicación será el mismo que en el ejemplo anterior.

Paso bidireccional de información entre actividades

Supón que ahora quieres devolver algo más, más información, ya bien sea mediante la intervención del usuario o generada por la propia *activity* llamada. Ahora añade una *RatingBar* a la **activity_second.xml**.

```
<RatingBar
    android:id="@+id/ratingBar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="32dp"
    android:numStars="5"
    android:rating="10"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/tv_nameReceived" />
```

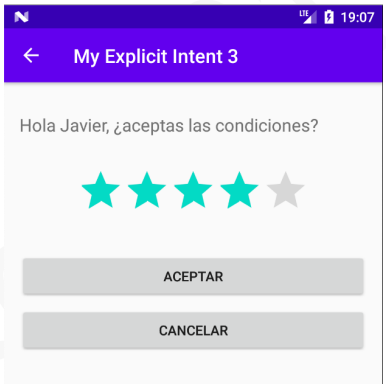


Figura 7

A continuación, modifica la clase `SecondActivity.kt` para que recoja el valor de la *RatingBar* y lo pase a la *activity* que ha realizado la llamada.

```
binding.btAccept.setOnClickListener {
    val intentResult : Intent = Intent().apply {
        // Se añade el valor del rating.
        putExtra(EXTRA_RESULT, binding.ratingBar.rating)
    }

    Log.d(TAG_APP, "Se devuleve valor de rating")
    setResult(Activity.RESULT_OK, intentResult)
    finish()
}
```

Únicamente, al pulsar el botón *Aceptar* se devolverá el resultado que el usuario haya establecido mediante la *RatingBar*. Se deberá crear un nuevo *Intent* al pulsar el botón, al que se le añadirá el resultado y el método `setResult()`, en este caso incluirá el *Intent* creado como parámetro. Ahora se deberá recoger desde la `MainActivity.kt` el resultado establecido.

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)

    if (requestCode == REQUEST_CODE) {

        if (resultCode == Activity.RESULT_OK) {
            val resultado = data?.getFloatExtra(EXTRA_RESULT, 0.0F).toString()

            binding.tvResult.text = "Condiciones aceptadas.\n" +
                    "Gracias por puntuar con $resultado estrellas."

            binding.tvResult.visibility = View.VISIBLE
        }

        if (resultCode == Activity.RESULT_CANCELED) {
            binding.tvResult.text = "Se canceló el contrato!"
            binding.tvResult.visibility = View.VISIBLE
        }
    }
}
```

En este caso se hace uso del parámetro `data` del método sobrecargado. Fíjate que se utiliza el operador `?`, esto es debido a que `data` puede ser nulo.

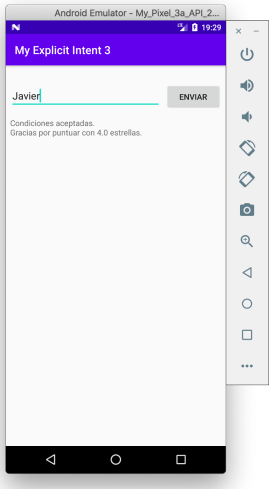


Figura 8

Control del onBackPressed()

Ahora que ya sabes abrir otras actividades, te encontrarás con algunos inconvenientes al utilizar el botón atrás del sistema. Para poder controlar sus acciones, deberás sobrecargar el método `onBackPressed()` de la actividad. Algo simple, como desactivarlo, se podría hacer comentando la llamada al constructor, quedando como se muestra a continuación.

Además, se añade que muestre un *Toast* al pulsarlo, pero puedes añadir todo aquello que necesites que haga según sea el caso.

```
override fun onBackPressed() {
    Toast.makeText(
        applicationContext,
        "onBackPressed",
        Toast.LENGTH_SHORT
    ).show()

    //super.onBackPressed()
}
```

5.3. Lanzar una tarea (*intent* implícito) y tratamiento de permisos

Cuando se quiere lanzar una tarea que no es propia de la *app* se tendrá que tratar con otro tema, los **permisos**. El tratamiento de los permisos en Android cambió a partir de la API 23, hasta entonces se concedían durante la instalación. Ahora se deben conceder de manera explícita, ya no se pide permiso durante el proceso de instalación, si no en tiempo de ejecución.

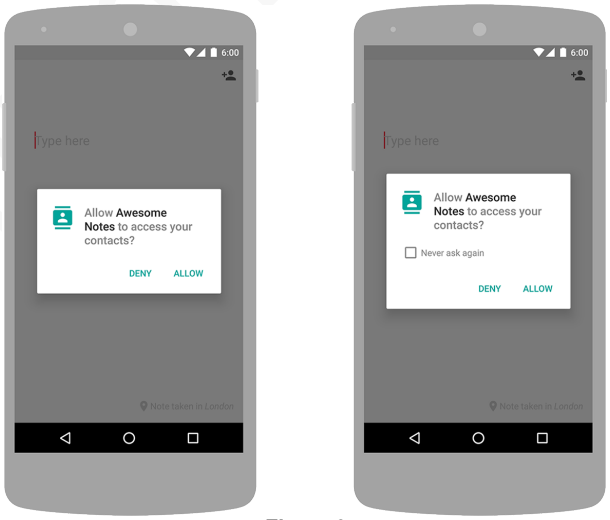


Figura 9

A raíz de este cambio se produce una clasificación de los permisos, básicamente se distinguirán tres tipos de permisos según el nivel de peligrosidad.

- **Permisos normales:** estos se utilizan cuando la aplicación necesita acceder a recursos o servicios fuera del ámbito de la *app*, donde no existe riesgo para la privacidad del usuario o para el funcionamiento de otras aplicaciones, por ejemplo, cambiar el uso horario.

  Si se declara en el *manifest* de la aplicación uno de estos permisos, el sistema otorgará automáticamente permiso para su uso durante la instalación. Además de no preguntarse al usuario por ellos, estos no podrán revocarlos.

- **Permisos de firma:** estos son concedidos durante la instalación de la *app*, pero sólo cuando la aplicación que intenta utilizar el permiso está firmada por el mismo certificado que la aplicación que define el permiso. Estos permisos no se suelen utilizar con aplicaciones de terceros, es decir, se utilizan entre aplicaciones del mismo desarrollador.

- **Permisos peligrosos:** estos permisos involucran áreas potencialmente peligrosas, como son la privacidad del usuario, la información almacenada por los usuarios o la interacción con otras aplicaciones. Si se declara la necesidad de uso de uno de estos permisos, se necesitará el consentimiento explícito del usuario, y se hará en tiempo de ejecución. Hasta que no se conceda el permiso, la *app* no podrá hacer uso de esa funcionalidad. Por ejemplo, acceder a los contactos.

Puedes encontrar todos los permisos que se pueden utilizar en la documentación de Google para Android[6]. El valor que necesites añadir al *manifest* lo encontrarás en *Constant Value*, y *Protection level* indica el tipo de permiso que es.

```
INTERNET

public static final String INTERNET

Allows applications to open network sockets.

Protection level: normal

Constant Value: "android.permission.INTERNET"
```

Comienza con un ejemplo sencillo en el que el único permiso que se necesite sea hacer uso de *Internet*, este es un **permiso normal**, por lo que se concederá el permiso durante la instalación de la aplicación.

Partiendo de un proyecto nuevo con API mínima 21 y una *empty activity*. Añade un botón en la *activity_main* para abrir una página web. A continuación, añade el permiso en el *manifest* de la aplicación.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="es.javiercarrasco.myimplicitintent">
4
```

6 *Manifest permission* (https://developer.android.com/reference/android/Manifest.permission)

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

    <application
        ...
```

El código **Kotlin** que necesita la *app* para lanzar un *intent* implícito, en este caso abrir una página web, será como se muestra a continuación. En el método `onCreate()` de la clase `MainActivity.kt` se añade la funcionalidad del botón.

```
// Abrir una página web.
binding.btnWebpage.setOnClickListener {
    val intent = Intent(
        Intent.ACTION_VIEW,
        Uri.parse("https://www.javiercarrasco.es")
    )

    if (intent.resolveActivity(packageManager) != null) {
        startActivity(intent)
    } else {
        Log.d("DEBUG", "Hay un problema para encontrar un navegador.")
    }
}
```

Al pulsar el botón se crea un *intent* que realizará la acción ACTION_VIEW[7], esta es la acción más común, se utiliza para que la información que se manda se muestre en la actividad. Por último se le indicará la URL utilizando la clase `Uri`[8] y su método `parse`.

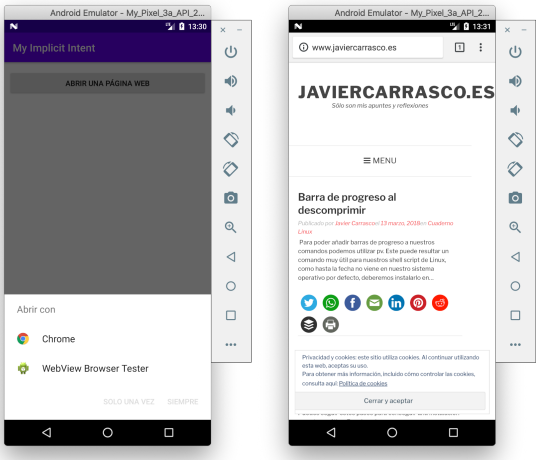


Figura 10

Cuando el usuario pulse el botón, si el sistema detecta más de una forma de cumplir la petición preguntará como debe resolverla, permitiendo dejarla por defecto si se elige la opción *SIEMPRE*.

7 *Action View* (https://developer.android.com/reference/kotlin/android/content/Intent.html#action_view)
8 Uri (https://developer.android.com/reference/android/net/Uri)

Fíjate en el uso de `resolveActivity()` antes de iniciar el *intent*, esto se hace para comprobar previamente que puede gestionarse la petición y evitar un posible fallo de la aplicación.

A continuación, amplia el ejemplo anterior mediante el uso de **permisos peligrosos**, por lo que deberás añadir gestión de permisos a la *app*. En primer lugar, añade un segundo botón a tu *activity_main* para poder realizar una llamada. Para que la *app* pueda realizar la llamada deberás añadir el siguiente permiso al *manifest*.

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

Si consultas el manual de referencia de Android, podrás ver que el permiso CALL_PHONE[9] está clasificado como peligroso.

Seguidamente, crea una constante en la actividad principal para contener el código de aceptación del permiso, este código será un número a nuestra elección.

```
// Constante que contiene el valor asignado al permiso de la app.
private val MY_PERMISSIONS_REQUEST_CALL_PHONE = 234
```

Ahora añade el evento sobre el nuevo botón para realizar la llamada, este será el momento en el que se deberá controlar el estado de los permisos y actuar en consecuencia.

```
// Realizar una llamada telefónica.
binding.btnCallphone.setOnClickListener {
```

Esta primera condición permitirá comprobar el estado de permiso en el contexto en el que se encuentre, `.checkSelfPermission()` devolverá verdadero si ya ha sido concedido o falso en caso contrario.

```
    // Se comprueba si el permiso en cuestión está concedido.
    if (ContextCompat.checkSelfPermission(this,
            Manifest.permission.CALL_PHONE)
        != PackageManager.PERMISSION_GRANTED
    ) {
```

Si entra es debido a que el permiso no está concedido, por lo que se deberá solicitar el de manera expresa al usuario.

```
        // Permiso no concedido.
        Log.d("DEBUG", "No está concedido el permiso para llamar")
```

El bloque ***if-else*** que viene a continuación puede tratarse como algo opcional, pero ayudará bastante al usuario, sobretodo, si es necesario dar una explicación de porqué se necesita que el permiso sea concedido. `.shouldShowRequestPermissionRationale()` devuelve verdadero si el usuario ya ha rechazado el permiso una vez, por lo que ya se podría mostrar una explicación al usuario.

```
        // Si el usuario ya ha rechazado al menos una vez (TRUE),
        // se da una explicación.
        if (ActivityCompat.shouldShowRequestPermissionRationale(
```

9 *Call Phone* (https://developer.android.com/reference/android/Manifest.permission.html#CALL_PHONE)

```
            this, Manifest.permission.CALL_PHONE)
        ) {
            Log.d("DEBUG", "Se da una explicación")
```

Las siguientes líneas permiten mostrar un cuadro de diálogo mediante la clase `AlertDialog.Builder`[10] (se verá en el siguiente capítulo). Fíjate en lo sencillo que resulta crear un cuadro de diálogo, la parte más compleja la puedes encontrar en la funcionalidad que quieras asignar a los botones.

```
            val builder = AlertDialog.Builder(this)
            builder.setTitle("Permiso para llamar")
            builder.setMessage("Puede resultar interesante indicar porqué.")

            // las variables dialog y which en este caso no se utilizan
            // se podrían sustituir por _ cada una ({_, _ -> ...).
            builder.setPositiveButton(android.R.string.ok) { dialog, which ->
                Log.d("DEBUG", "Se acepta y se vuelve a pedir permiso")

                ActivityCompat.requestPermissions(
                    this, arrayOf(Manifest.permission.CALL_PHONE),
                    MY_PERMISSIONS_REQUEST_CALL_PHONE
                )
            }
            builder.setNeutralButton(android.R.string.cancel, null)
            builder.show()
```

En el ***else*** se realizará la petición de concesión del permiso si no hay que dar explicación. El método `requestPermissions()`[11] de la clase `ActivityCompat` se encargará de lanzar la petición de aceptación del permiso.

```
        } else {
            // No requiere explicación, se pregunta por el permiso.
            Log.d("DEBUG", "No se da una explicación.")
            ActivityCompat.requestPermissions(
                this, arrayOf(Manifest.permission.CALL_PHONE),
                MY_PERMISSIONS_REQUEST_CALL_PHONE
            )
        }
```

Por último, se entrará en el siguiente ***else*** si el permiso ya ha sido concedido, por lo que se creará la llamada al *intent* para realizar la llamada telefónica mediante la aplicación predefinida del sistema.

```
    }else{
        Log.d("DEBUG", "El permiso ya está concedido.")
        val intent = Intent(
            Intent.ACTION_CALL,
            Uri.parse("tel:965555555")
        )
```

10 AlertDialog (https://developer.android.com/reference/kotlin/android/app/AlertDialog.Builder)
11 requestPermissions (https://developer.android.com/reference/kotlin/androidx/core/app/ActivityCompat)

```
48          startActivity(intent)
49      }
50  } // Fin btnCallphone.setOnClickListener
```

Llegado a este punto, la *app* está lista para solicitar permiso para llamar por teléfono. Ahora, si se necesita recoger la respuesta del usuario sobre el método `requestPermissions()` se deberá sobrecargar el siguiente método.

```
1   // Se analiza la respuesta del usuario a la petición
2   // de permisos.
3   override fun onRequestPermissionsResult(
4       requestCode: Int,
5       permissions: Array<out String>,
6       grantResults: IntArray
7   ) {
8       super.onRequestPermissionsResult(requestCode, permissions, grantResults)
9
10      when (requestCode) {
11          MY_PERMISSIONS_REQUEST_CALL_PHONE -> {
12              if ((grantResults.isNotEmpty() && grantResults[0]
13                          == PackageManager.PERMISSION_GRANTED)) {
14                  Log.d("DEBUG", "Permiso concedido!!")
15              } else {
16                  Log.d("DEBUG", "Permiso rechazado!!")
17              }
18              return
19          }
20          else -> {
21              Log.d("DEBUG", "Se pasa de los permisos.")
22          }
23      }
24  }
```

Observa el resultado de todo este código al pulsar el botón para realizar una llamada telefónica por primera vez, ésta sería la primera vez que se solicita permiso al usuario.

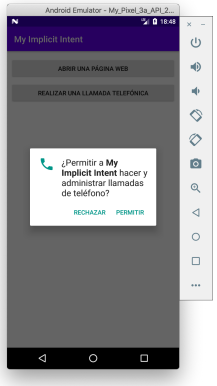


Figura 11

Si se rechaza la primera vez que aparece el aviso, la siguiente vez que se pulse el botón se mostrará la explicación al usuario y, se actuará según la opción pulsada en el *AlertDialog*.

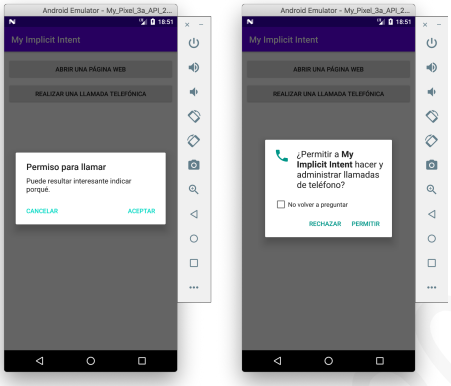


Figura 12

Ya se ha visto como comprobar el estado de un permiso, en este caso llamar por teléfono, pero si se necesita comprobar más de uno, se deberá modificar ligeramente la forma de afrontar esta operación, evidentemente, existe más de una forma de enfrentarse a esta situación.

A continuación, se añadirá a la actividad principal un tercer botón, por ejemplo, para poder hacer una foto, como aparece reflejado en la figura.

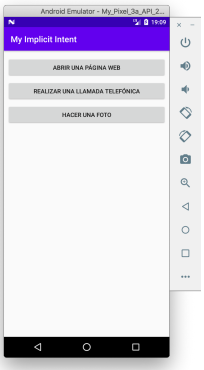


Figura 13

Ahora, deberás modificar el *manifest* del proyecto para añadir los nuevos permisos que se deberán controlar en la desde la aplicación y, sobre los que se deberá solicitar permiso.

```
1   <uses-permission android:name="android.permission.INTERNET" />
2   <uses-permission android:name="android.permission.CALL_PHONE" />
3   <uses-permission android:name="android.permission.CAMERA" />
```

Recuerda que el acceso a *Internet* no está considerado peligroso, por lo que no es necesario pedir al usuario permiso de manera explícita. A continuación, se creará una nueva clase **Kotlin** que se llamará `GestionPermisos` (***File > New > Kotlin File/Class***), la cual se encargará de la gestión de los permisos tal como se ha visto en el punto anterior, pero de una manera más organizada.

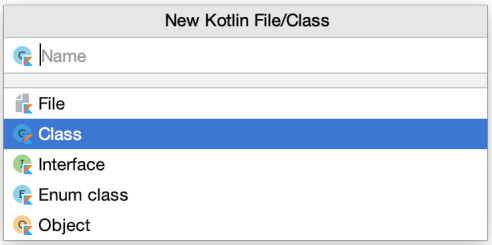


Figura 14

```
class GestionPermisos(
    val activity: Activity,
    val permiso: String,
    val code: Int
) {
    fun checkPermissions(): Boolean {
        // Se comprueba si el permiso en cuestión está concedido.
        if (ContextCompat.checkSelfPermission(
                activity, permiso
            ) != PackageManager.PERMISSION_GRANTED
        ) {
            // Si no está concedido el permiso se entra.
            Log.d("DEBUG", "No tienes permiso para esta acción: $permiso")

            // Si el usuario ya lo ha rechazado al menos una vez (TRUE),
            // se puede mostrar una explicación.
            if (ActivityCompat.shouldShowRequestPermissionRationale(
                    activity, permiso)
            ) {
                Log.d("DEBUG", "Se da una explicación")
                showAlert()
            } else {
                // No requiere explicación, se pregunta por el permiso.
                Log.d("DEBUG", "No se da una explicación.")
                ActivityCompat.requestPermissions(
                    activity, arrayOf(permiso), code
                )
            }
        } else {
            Log.d("DEBUG", "Permiso ($permiso) concedido!")
        }

        return ContextCompat.checkSelfPermission(
            activity, permiso
        ) == PackageManager.PERMISSION_GRANTED
    }

    // Función encargada de mostrar un AlertDialog con información adicional.
    private fun showAlert() {
        val builder = AlertDialog.Builder(activity)

        builder.setTitle("Concesión de permisos")
        builder.setMessage(
            "Puede resultar interesante indicar" +
                    " porqué se necesita conceder permiso."
        )

        builder.setPositiveButton(android.R.string.ok) { _, _ ->
            Log.d("DEBUG", "Se acepta y se vuelve a pedir permiso")
            ActivityCompat.requestPermissions(
```

```
                activity, arrayOf(permiso), code
            )
        }

        builder.setNeutralButton(android.R.string.cancel, null)
        builder.show()
    }
}
```

Esta clase tiene tres parámetros que se deberán pasar para poder comprobar un permiso en cuestión, *activity* para indicar el contexto sobre el que se está trabajando, *permiso* para indicar el permiso que se quiere comprobar, y *code* para indicar nuestro código de aceptación del permiso. En este caso, al tener más de un permiso que aceptar, en la clase principal, se le cambia el nombre a la variable `MY_PERMISSIONS_REQUEST_CALL_PHONE` por `MY_PERMISSIONS_REQUEST_CODE`, la función será la misma. También se añadirá `private lateinit var gestionPermisos: GestionPermisos`, con estos cambios, la clase `MainActivity.kt` quedará de la siguiente forma.

```
class MainActivity : AppCompatActivity() {
    // Constante que contiene el valor asignado al permiso de la app.
    private val MY_PERMISSIONS_REQUEST_CODE = 234

    // ViewBinding
    private lateinit var binding: ActivityMainBinding

    private lateinit var gestionPermisos: GestionPermisos

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        // Abrir una página web.
        binding.btnWebpage.setOnClickListener {
            val intent = Intent(
                    Intent.ACTION_VIEW,
                    Uri.parse("http://www.javiercarrasco.es")
            )
            startActivity(intent)
        }

        // Realizar una llamada telefónica.
        binding.btnCallphone.setOnClickListener {
            // Se comprueba si el permiso en cuestión está concedido.
            if (ContextCompat.checkSelfPermission(
                    this, Manifest.permission.CALL_PHONE
                ) == PackageManager.PERMISSION_GRANTED) {
                Log.d("DEBUG", "El permiso ya está concedido.")

                val intent = Intent(
                    Intent.ACTION_CALL,
```

```
                    Uri.parse("tel:965555555")
                )
                startActivity(intent)
            }else{
                gestionPermisos = GestionPermisos(this,
                    Manifest.permission.CALL_PHONE,
                    MY_PERMISSIONS_REQUEST_CODE)
                gestionPermisos.checkPermissions()
            }
        }

        // Abrir la cámara de fotos.
        binding.btnTakepicture.setOnClickListener {
            // Se comprueba si el permiso en cuestión está concedido.
            if (ContextCompat.checkSelfPermission(
                    this, Manifest.permission.CAMERA
                ) == PackageManager.PERMISSION_GRANTED) {
                Log.d("DEBUG", "El permiso ya está concedido.")

                val intent = Intent(Intent(MediaStore.ACTION_IMAGE_CAPTURE))
                startActivity(intent)
            }else{
                gestionPermisos = GestionPermisos(this,
                    Manifest.permission.CAMERA,
                    MY_PERMISSIONS_REQUEST_CODE)
                gestionPermisos.checkPermissions()
            }
        }
    }

    // Se analiza la respuesta del usuario a la petición de permisos.
    override fun onRequestPermissionsResult(
        requestCode: Int,
        permissions: Array<out String>,
        grantResults: IntArray
    ) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults)

        when (requestCode) {
            MY_PERMISSIONS_REQUEST_CODE -> {
                Log.d("DEBUG", "${grantResults[0]} ${permissions[0]}")

                if ((grantResults.isNotEmpty() && grantResults[0]
                            == PackageManager.PERMISSION_GRANTED)) {
                    Log.d("DEBUG", "Permiso concedido!!")
                } else {
                    Log.d("DEBUG", "Permiso rechazado!!")
                }
                return
            }
```

```
            else -> {
                Log.d("DEBUG", "Se pasa de los permisos.")
            }
        }
    }
}
```

Fíjate que se sigue comprobando si el permiso está concedido mediante el método `checkSelfPermission()`, esto se hace porque seguidamente lanzamos un *intent* y Android no permite lanzar una nueva actividad si se necesita la concesión de un permiso peligroso.

Observa también que el método `onRequestPermissionsResult()` no ha variado con respecto al ejemplo anterior, ya que se utiliza el mismo código de respuesta para todos los permisos, pero, si se quisiese tratar la respuesta a cada permiso, se deberá crear un código diferente para cada uno.

También se podría afrontar la gestión de permisos lanzando una comprobación inicial, por ejemplo, creando una lista con los permisos a evaluar.

```
// Lista de permisos necesarios.
val permissionsList = listOf<String>(
    Manifest.permission.INTERNET,
    Manifest.permission.CALL_PHONE,
    Manifest.permission.CAMERA,
    Manifest.permission.READ_CALENDAR
)
```

5.3.1. Ejemplos básicos de *intents* implícitos

Además de los *intents* implícitos ya vistos, otros que podrían resultar de utilidad serían los que se muestran a continuación.

Añadir una alarma al despertador

Para este case es necesario establecer el permiso correspondiente para poder crear una alarma en el despertador, en el *manifest* añade la siguiente línea.

```
<uses-permission android:name="com.android.alarm.permission.SET_ALARM" />
```

El código que permite realizar esta acción será el siguiente que se muestra.

```
val intent = Intent(AlarmClock.ACTION_SET_ALARM)
    .putExtra(AlarmClock.EXTRA_MESSAGE, "Se acabó dormir")
    .putExtra(AlarmClock.EXTRA_HOUR, 7)
    .putExtra(AlarmClock.EXTRA_MINUTES, 45)

if (intent.resolveActivity(packageManager) != null) {
    startActivity(intent)
}
```

Mandar un SMS

El siguiente código permite enviar un SMS mediante la aplicación del teléfono, por lo que no se necesita conceder permisos en la aplicación.

```
val intent = Intent(
    Intent.ACTION_SENDTO,
    Uri.parse("smsto:" + 777666777)
)

intent.putExtra("sms_body", "Cuerpo del mensaje")

if (intent.resolveActivity(packageManager) != null) {
    startActivity(intent)
}
```

Fíjate como se indica el destinatario mediante el método `Uri` y como se añade texto al cuerpo del mensaje haciendo uso del método `putExtra()`.

Mandar un correo electrónico y compartir información

Este ejemplo trata de compartir información con ese tipo de aplicaciones que así lo especifiquen mediante los ***intent-filter***, en ese caso se tratará de enviar un correo electrónico, y se verá como el correo electrónico aparece completado a falta de enviar mediante la aplicación, esto hace que no sea necesario indicar un permiso específico para ello en nuestra aplicación.

```
val TO = arrayOf("javier@javiercarrasco.es")
val CC = arrayOf("")
val intent = Intent(Intent.ACTION_SEND)

intent.type = "text/html" // o también text/plain
intent.putExtra(Intent.EXTRA_EMAIL, TO)
intent.putExtra(Intent.EXTRA_CC, CC)
intent.putExtra(Intent.EXTRA_SUBJECT, "Envío de un email desde Kotlin")
intent.putExtra(Intent.EXTRA_TEXT, "Esta es mi prueba de envío de un correo.")

if (intent.resolveActivity(packageManager) != null) {
    startActivity(Intent.createChooser(intent, "Enviar correo..."))
}
```

Los campos para `EXTRA_EMAIL` y `EXTRA_CC` deben crearse mediante el uso de *arrayOf*, esto es debido a que se puede especificar más de un correo. El campo *type* determinará el tipo de aplicación que se utilizará para enviar el correo, cuando el sistema pregunta, suele ofrecer más opciones si se trata texto plano.

Hacer una foto y recuperar el resultado

El siguiente ejemplo muestra cómo lanzar un *intent* hacia la cámara de fotos y obtener el resultado de la acción para mostrarlo en un *ImageView*.

Para este ejemplo se utilizará el siguiente permiso en el fichero manifest, además, se marcará como necesario para poder ejecutarse la acción.

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

Tras comprobar si está o no concedido el permiso, el código para lanzar el *intent* será el siguiente.

```
val intent = Intent(Intent(MediaStore.ACTION_IMAGE_CAPTURE))

if (intent.resolveActivity(packageManager) != null) {
    startActivityForResult(intent, MY_PERMISSIONS_REQUEST_CODE)
}
```

Tras lanzar el *intent*, deberá sobrecargase el método `onActivityResult()` para poder capturar el resultado obtenido y cargar el *thumbnail* en un *ImageView*.

```
// Se recupera la imagen capturada.
override fun onActivityResult(requestCode: Int,
    resultCode: Int, data: Intent?) {

    super.onActivityResult(requestCode, resultCode, data)
    if (requestCode === MY_PERMISSIONS_REQUEST_CODE && resultCode === RESULT_OK)
    {
        val thumbnail: Bitmap = data?.getParcelableExtra("data")!!
        binding.imageView.setImageBitmap(thumbnail)
    }
}
```

Pero, deberás tener en cuenta que desde Android X, `startActivityForResult()` y `onActivityResult()` han sido marcados como *deprecated*, por lo que recomiendan hacer uso del método `registerForActivityResult()`, cambiando la forma de trabajar con *intents*.

5.4. Vista horizontal

Llegados este punto, la *app* de ejemplo ya dispone de varios botones y, si se gira el dispositivo, poniéndolo en horizontal, estos se adaptarán a la pantalla.

Pero puede darse el caso de tener más botones, y tal vez, esta disposición no sea deseable, por lo que se puede diseñar una nueva pantalla para cuando el dispositivo cambie de posición. Para ello, abre el *layout* que quieras adaptar y selecciona la opción ***Create Landscape Variation*** tal como se muestra la figura.

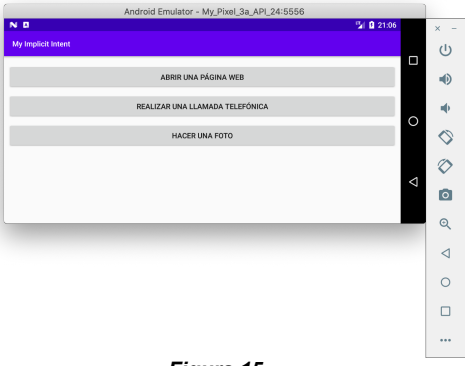
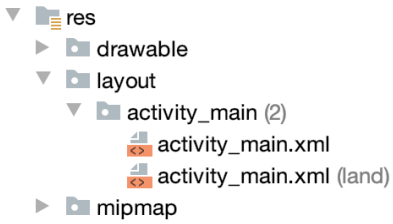


Figura 15

Ahora, podrás observar cómo se dispondrá de dos archivos *XML* con el mismo nombre, pero, diferenciados por el indicador *(land)* en el explorador de proyecto. Recuerda que aunque tengan el mismo nombre, deberás editarlos por separado.



Por ejemplo, si se añadiese un cuarto botón al ejemplo que has desarrollado en este apartado, podría ser interesante adaptar la vista horizontal de la siguiente forma para no tener unos botones tan largos.

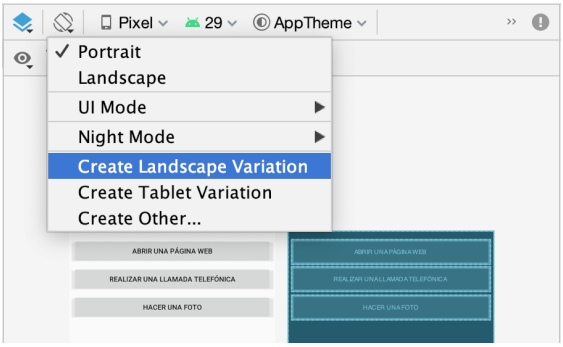


Figura 16

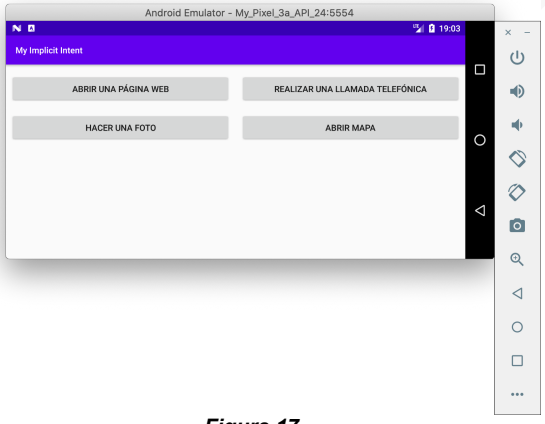


Figura 17

El cuarto botón añadido permitirá abrir la aplicación Google Maps[12] con una ubicación predeterminada, más adelante se verá como trabajar con mapas y con la ubicación en tiempo real.

```
// Abrir Google Maps.
binding.btnOpenmap.setOnClickListener {
    // Para abrir Google Maps, si no se requiere ubicación, no es
    // necesario solicitar permiso.
    val intent = Intent(
        Intent.ACTION_VIEW,
        Uri.parse("geo:0,0?q=Alicante")
    )

    startActivity(intent)
}
```

12 Abrir un mapa (https://developer.android.com/guide/components/intents-common#ViewMap)

Supongamos ahora, que se añade un quinto botón a la vista horizontal, pero que no está presente en la vista vertical. Siguiendo el esquema visto, seguramente la aplicación producirá un error, pero *Kotlin* ofrece una serie de herramientas para solventar este tipo de problemas.

En primer lugar, se utilizará el operador `?` para indicar la posibilidad de encontrar un nulo en el elemento en cuestión, y seguidamente el método `let` para que, en caso de no ser nulo, indicar que debe hacer. Con un quinto botón en la vista horizontal, la *activity* mostrará dos UI en la aplicación según la posición del dispositivo.

Esta es una opción muy buena a tener en cuenta, ya que cuando se crean diferentes vistas para un mismo *layout*, como es el caso de los dos *layouts* para `activity_main.xml`, se sigue disponiendo una una sola clase para gestionarlos, de ahí que deba controlarse la existencia de elementos presentes en una vista pero no en la otra.

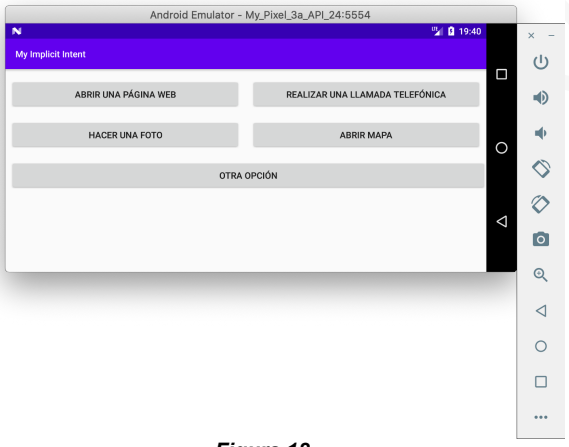


Figura 18

El código *Kotlin* adaptado a esta nueva situación de la aplicación puede ser como el que se muestra a continuación.

```
// Botón otras opciones.
binding.btnOtheroption?.let {
    it.setOnClickListener {
        Toast.makeText(
            this,
            "Pulsado el botón \"${getString(R.string.button_otheroption)}\".",
            Toast.LENGTH_LONG
        ).show()
    }
}
```

Ahora, la aplicación mostrará cuatro botones en la vista vertical y cinco cuando cambie de posición, mostrando la vista horizontal.

Por último, puede resultar útil conocer en un momento dado en que posición se encuentra el dispositivo móvil. Para obtener esta información se puede utilizar la siguiente línea.

```
val estado = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.R) {
    this.display?.rotation
} else {
    @Suppress("DEPRECATION")
    this.windowManager.defaultDisplay.rotation
}
```

Se debe utilizar el chequeo de versión debido a que el uso de defaultDisplay está marcado como deprecad desde la API 30 (Android R).

En este caso, la variable `estado` podría obtener los siguientes valores según la posición del dispositivo. Se tendrá como referencia la posición del auricular del dispositivo o la cámara frontal.

- **0** en vertical a 90º (auricular en la parte superior).
- **1** en horizontal a 180º.
- **2** en vertical a 270º.
- **3** horizontal a 0º o 360º.

Ejercicios propuestos

5.1. Crea una aplicación **Android+Kotlin** llamada ***My First Intent***, con API mínima 21. La *app* pedirá en su primera *activity* que escribas una pregunta en un *EditText* y, mediante un botón enviar, deberá mandar la pregunta a una segunda *activity*.

La segunda *activity* mostrará la pregunta en un *TextView*, además, tendrá un *EditText* en el cual se deberá escribir la respuesta, junto con un botón que devolverá el resultado de la respuesta a la primera *activity*.

Para terminar, la primera *activity* mostrará la respuesta en un *TextView*.

5.2. Termina el ejemplo visto para abrir *intents* implícitas y gestión de permisos. Añade la posibilidad de abrir *Google Maps* con la ubicación del Castillo de Santa Bárbara de Alicante, abrir el calendario y abrir el correo.

5.3. Recupera el ejercicio **4.10.3.** (capítulo 4) y modifica el código para que en lugar de mostrar un *Snackbar* al pulsar un *item*, se abra una nueva actividad (pantalla) con el nombre del elemento, su nombre en latín y la imagen.