Ejercicio: Intent

Tipos de comportamiento

Al iniciar un componente, podemos tener:

- **Intents Explícitos**: Cuando un intent se crea para enviar un mensaje a un componente específico se le llama Intent explícito. Por ejemplo, cuando iniciamos otra actividad y conocemos cual es exactamente la nueva actividad. Lo que significa que se hizo explicita la asociación del Intent a esta acción.
- **Intents Implicitos**: Un intent se comporta implícitamente cuando no se sabe con exactitud que componente se iniciará para recibir el mensaje que vamos a enviar. Aunque no se sabe específicamente, si se tiene claro cuál debe ser el propósito del componente que sea iniciado.

¿Cómo Iniciar Una Actividad Desde Otra Actividad?

```
val intent = Intent( packageContext: this, ActividadNueva::class.java)
startActivity(intent)
```

Para invocar a otra actividad hemos usado **Intent**. Ten en cuenta que Intent, representa la «**intención**» de enviar un mensaje a otro componente, que en este caso es iniciar una actividad. El constructor de intent recibe dos parámetros. El primero es el contexto desde deseamos enviar el mensaje, en este caso es nuestra actividad principal. El segundo hace referencia a la clase del componente receptor del mensaje que será nuestra clase ActividadNueva.

Los Intents permiten que enviemos datos al enlazar dos actividades en nuestra aplicación. A estos datos se les denominan ***Extras*** y se componen de un **identificador** y un **valor.** Antes de iniciar la actividad debemos adherir los datos al intent con el método putExtra().

```
val intent = Intent( packageContext: this, ActividadNueva::class.java)
intent.putExtra(EXTRA_TELEFONO, binding.telefono.text.toString())
intent.putExtra(EXTRA_MENSAJE,  value: "hola")
startActivity(intent)
```

Desde la actividad destino, ¿Cómo recibimos estos datos "extra"? Desde el método onCreate, invocaremos el método getIntent() de la clase Activity y obtendremos el intent que inicio la actividad. Todas las actividades son iniciadas por un intent, incluso podemos obtener el Intent de la actividad principal iniciado por el sistema.

Tras tener el intent, extraeremos la cadena del mensaje que enviamos con getStringExtra().

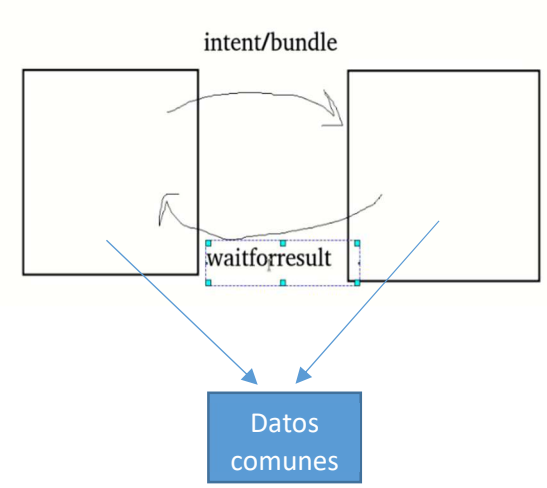
```
//Obteniendo la instancia del Intent
val intent = intent

//Extrayendo el extra de tipo cadena
val mensaje = intent.getStringExtra(MainActivity.EXTRA_MENSAJE)
val telefono = intent.getStringExtra(MainActivity.EXTRA_TELEFONO)
```

Pasar información entre Activities II

Podemos pasar información entre activitys como hemos visto en el caso anterior, o bien utilizando una clase tipo Object que implemente el interfaz java.io.Serializable, ya que las activitys comparten memoria.

En la siguiente figura podemos hacernos una idea, en la parte de arriba aparecen los mecanismos para pasar datos entre ventanas, abajo se ve la zona de memoria compartida que son accesibles desde todas las activitys:



Diferencias clave entre Intent y Bundle:

- Intent describe una acción a realizar y puede incluir datos adicionales (como un Bundle).
- Bundle es solo un contenedor de datos que se puede utilizar para empaquetar información que puede ser pasada junto con un Intent u otras estructuras (como el estado de una actividad).

En resumen, los Intent son para iniciar acciones o comunicaciones entre componentes, mientras que los Bundle se usan para almacenar y transmitir datos a través de esos Intent.

Crea ahora un paquete modelo y dentro una clase de datos que se llame **Persona** que tenga los campos nombre y edad e implemente la interfaz *java.io.Serializable*.

En lugar de hacer Serializable, para Android se recomienda hacer Parcelable ya que es más eficiente para Android, consumiendo menos memoria y siendo más rápido el paso del objeto, pero hay que hacer varios retoques en Graddle y varios import y no nos merece la pena, porque veremos cómo pasar los datos de otras maneras.

```
//Paso de datos usando Bundle
var persona = Persona(binding.edNombre.text.toString(),
binding.edEdad.text.toString().toInt())

val bundle = Bundle()
bundle.putString("nombre1", binding.edNombre.text.toString())
bundle.putSerializable("persona",persona)


var intent: Intent = Intent(this, Ventana2::class.java)
intent.putExtra("datos",bundle)
startActivity(intent)
```

Para recuperar los datos en la activitity llamada Ventana2 con Bundle

```
val bundle = intent.getBundleExtra("datos")
val nombre = bundle?.getString("nombre1")
val persona = bundle?.getSerializable("persona" , Persona::class.java)
```

EJERCICIO: Probar lo aprendido creando las siguientes ventanas:

1. Una manera de reiniciar un activity, descargando todo y volviendo a empezar. Pruébalo en tu aplicación:

```
binding.btReiniciar.setOnClickListener {
    var ine : Intent = intent
    finish()
    startActivity(ine)
}
```

2. Ahora veremos como llamar a una ventana2 y esperar que esa ventana nos devuelva resultados

Ventana inicial

```
//Se necesita definir una variable (resultlancher) definida como
atributo de la clase (ver más adelante).

binding.btEsperarRespuestaActual.setOnClickListener {

    // Start the SecondActivity
    val intent = Intent(this, Ventana2::class.java)
    resultLauncher.launch(intent)
}
```

Ventana 2

```
//Devolver datos a la ventana 1
binding.btDevolverActual.setOnClickListener {

    // Get the text from the EditText
    val stringToPassBack = binding.edDevolver.text.toString()

    // Put the String to pass back into an Intent and close this activity
    val intent = Intent()
    intent.putExtra("valorEdicionV2", stringToPassBack)
    setResult(Activity.RESULT_OK, intent)

    finish()
}
```

Y ahora al volver a la ventana que llama, se evalúa mediante una variable con lambda que has tenido que declarar previamente.

Ventana Inicial

```
//Esta variable es necesaria para la llamada y espera de forma
actual.

var resultLauncher =
registerForActivityResult(ActivityResultContracts.StartActivityFor
Result()) { result ->

    if (result.resultCode == Activity.RESULT_OK) {

        // There are no request codes
        val data: Intent? = result.data

        // Get String data from Intent
        val returnString = data!!.getStringExtra("valorEdicionV2")

        // Set text view with string
        binding.txtValorDevuelto.text = returnString
    }
}
```

Así de esta manera nos comunicaremos habitualmente con otras ventanas, muchas veces de servicios ya predefinidos de Google, como maps, permisos, etc es cogerle el truco

3. Otra manera de comunicar datos es mediante una clase Object visible por todos los activitys.

```
object Almacen {

    var listaPersonas =ArrayList<Persona>()

    fun addPersona(p:Persona){

        listaPersonas.add(p)

    }

    fun getPersonas():ArrayList<Persona>{

        return listaPersonas

    }
}
```

En la **Ventana principal** lo rellenamos así:

```
//ahora con object
Almacen.addPersona(p)
startActivity(inte)
```

Y en la **Ventana 2** lo recogemos así:

```
//ahora mediante el almacen
binding.txtCaja2.text = Almacen.getPersonas().toString()
```

Iniciar un navegador web Android desde nuestra actividad

A continuación, vamos a ejecutar una aplicación de navegación web desde nuestra actividad principal.

```
val webpage:Uri = Uri.parse( uriString: "https://www.google.es")
val webIntent:Intent = Intent(Intent.ACTION_VIEW, webpage)
startActivity(webIntent)
```

Usamos la constante Intent.ACTION_VIEW que representa la **visualización de contenido**, en este caso una página web asociada a un esquema de datos URI. Sin embargo, hay muchas más constantes asignadas a otras acciones estándar de **Android**.

A continuación, se presenta una tabla descriptiva sobre cada acción para que las tengas en cuenta:

Acción	Constante representativa
Empieza la actividad de una aplicación con el punto principal de entrada	ACTION_MAIN
Muestra información al usuario	ACTION_VIEW
Indica que cierta informacion debe ser añadida en una parte especifica	ACTION_ATTACH_DATA
Obtiene un acceso a la edicion de alguna informacion	ACTION_EDIT
Selecciona un item de un conjunto de datos y retorna en él	ACTION_PICK
Visualiza una actividad con opciones para el usuario.	ACTION_CHOOSER
Permite seleccionar un tipo de informacion y retornarla	ACTION_GET_CONTENT
Marca un numero para iniciar una llamada	ACTION_DIAL
Realiza una llamada a un numero especificado	ACTION_CALL
Inicia una accion de envio de datos a alguien que aun no se ha especificado	ACTION_SEND
Inicia una accion de envio de datos para alguien en especifico	ACTION_SENDTO
Gestiona una llamada entrante	ACTION_ANSWER
Crea un objeto vacio que se añadirá a un contenedor	ACTION_INSERT
Elimina informacion seleccionada de su ubicación	ACTION_DELETE
Ejecuta cualquier tipo de dato	ACTION_RUN
Inicia una sincronizacion de datos	ACTION_SYNC
Retorna en el nombre de la clase seleccionada	ACTION_PICK_ACTIVITY
Realiza una busqueda	ACTION_SEARCH
Realiza una busqueda web	ACTION_WEB_SEARCH
Ejecuta un punto de entrada principal para un teste en modo «test de fabrica»	ACTION_FACTORY_TEST

Todas estas acciones difieren en su forma de ejecución. No todas requieren que usemos URIs para activarlas, hay otras que necesitan que usemos el método putExtra() para ingresar parámetros necesarios para el funcionamiento.

Es posible que en el dispositivo que ejecutemos este intent no hayan aplicaciones que muestren contenido web. Si es así, la aplicación presentaría un error en tiempo de ejecución. Para evitar esta situación podemos usar el siguiente código:

```
val webIntent = Intent(Intent.ACTION_VIEW, Uri.parse( uriString: "https://www.google.es"))
// Verificar si hay aplicaciones disponibles
val activities = this.packageManager.queryIntentActivities(intent, PackageManager.MATCH_DEFAULT_ONLY)
val isIntentSafe = activities.size > 0
if (isIntentSafe) {
    startActivity(webIntent)
}
```

PackageManager puede proporcionarnos aquellos elementos que respondieron al intent con el método queryIntentActivities(). Luego comprobamos si al menos uno fue retornado, si fue así entonces iniciamos la actividad.

Ejemplos como utilizar Intents

https://developer.android.com/guide/components/intents-common?hl=es-419