Comunicación entre Activity y Fragment

Hay veces en que necesitamos que nuestra activity realice algo, ya sea para controlar y centralizar la lógica o porque hay distintas funciones que los fragments no pueden hacer. Por ello debemos entender que la forma más sencilla que tenemos de hacer esto es a través de los **listeners**.

Un **listener** no es más que una función que se llama en un sitio, pero avisa a otro de que ha sido llamado. Es decir, el listener hará de comunicador entre **activity** y **fragment** (aunque se puede utilizar en muchísimos más casos).

Para ello lo primero que hay que hacer es crear una **Clase Interface** a través de **new>Kotlin File Class** y nos saldrá un diálogo para configurar nuestro fichero.

La interfaz fija un contrato entre quien los usa. Si añadimos la interfaz al activity, todos los métodos que estén en la interfaz deberán añadirse a dicho activity.

En la interfaz solo se define la función con sus parámetros de entrada o salida si tuvieran, pero no la lógica, pues esta se tiene que implementar donde implementemos la interfaz (la activity).

```
1 interface OnFragmentActionsListener {
2     fun onClickFragmentButton()
3 }
```

Hemos añadido una función llamada **onClickFragmentButton()** que no recibe parámetros ni devuelve nada. Ahora implementaremos esta interfaz en la clase **MainActivity**. Para ello lo añadiremos en la primera línea, quedando nuestra activity así:

```
1
2 class MainActivity : AppCompatActivity(), OnFragmentActionsListe
3
      override fun onCreate(savedInstanceState: Bundle?) {
```

```
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }

}
```

Este código dará un error ya que no hemos añadido los métodos de la interface al MainActivity, para ello, iremos al menú superior **Code>Implement methods** y nos saldrá un diálogo con todos los métodos que debemos implementar.

Dentro del método insertado, se creará la lógica de funcionamiento de dicho método.

Pasamos ahora a definir la comunicación en el Fragment. Lo primero que haremos será añadir nuestro **listener** al fragment, para ello empezaremos declarando una variable **listener**, que sea del mismo tipo que la interfaz.

```
private var listener: OnFragmentActionsListener? = null
```

Esta variable la pondremos al inicio de la clase, y podrá ser **null**, así nos obligamos a comprobar su contenido antes de intentar ejecutar cualquier acción.

Luego para inicializarla usaremos uno de los métodos del ciclo de vida, la función **onAttach()**.

```
override fun onAttach(context: Context) {
    super.onAttach(context)
    if (context is OnFragmentActionsListener) {
        listener = context
    }
}
```

La primera línea llama a la función **super()** porque se desea añadir funcionalidad al método, no anular lo que haría normalmente.

Luego encontramos un **if()**, que comprobará si el contexto que llega a la función **onAttach()**, tiene implementada la interfaz que hemos creado. En este caso el contexto será de **MainActivity**, pues es esta clase la que crea el **fragment**. Para finalizar igualamos el **listener** que hemos declarado en la parte superior de la clase al contexto.

El último paso será que haremos será desconectar el fragment de la activity cuando vaya a morir. Usaremos otro método del ciclo de vida, **onDetach()**, volviéndolo nulo.

```
override fun onDetach() {
    super.onDetach()
    listener = null
}
```

Para llamar al **listener**, lo habitual es crear un **onClickListener()** para que cuando un botón del fragment se ejecute lo llame. Asignaremos el **onClick** en el método **onViewCreated** que como verás también forma parte del ciclo de vida.

```
override fun onViewCreated(view: View, savedInstanceState: Bundl
    super.onViewCreated(view, savedInstanceState)
    botonPulsado.setOnClickListener { listener?.onClickFragmentB
}
```

De esta manera, hemos hecho que al hacer clic en un botón, se llame a la función **onClickFragmentButton()** de la interfaz, que hará que nuestra **activity** ejecute la lógica del método asociado.