CFGS DAM

IES MONTE NARANCO

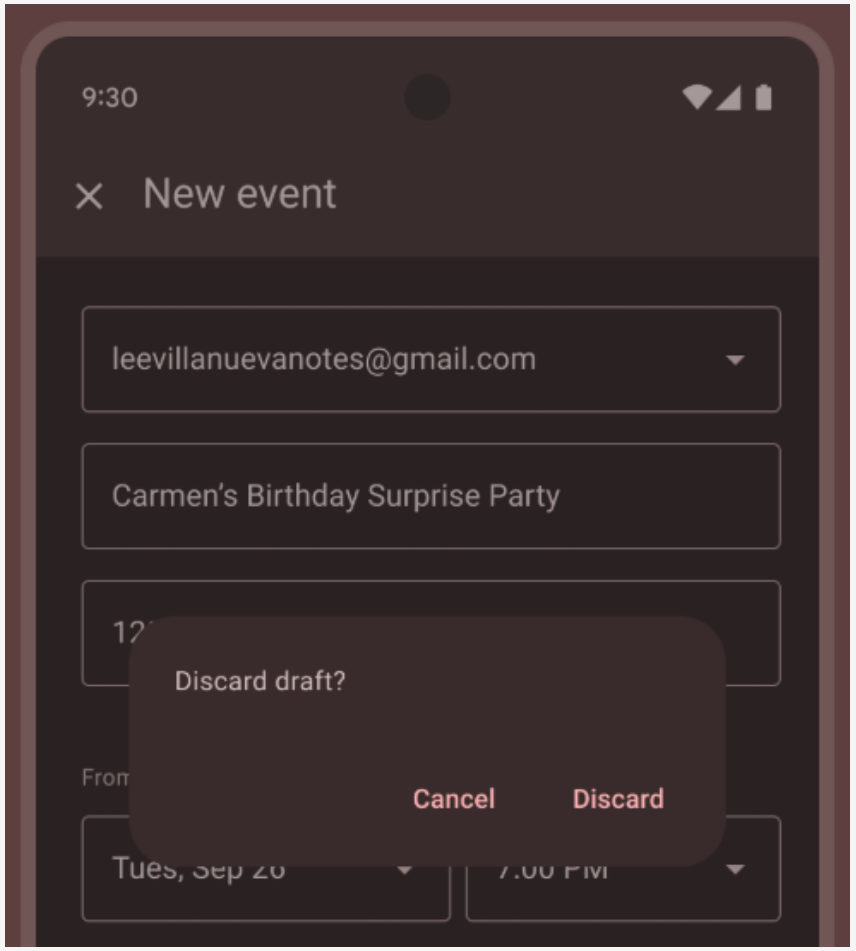
UT-4

Programación de aplicaciones Android

Diálogos

Diálogos

- **Diálogo:** ventana pequeña que le indica al usuario que debe tomar una decisión o ingresar información adicional.
- No ocupa toda la pantalla.
- Se suele usar para eventos modales que requieren que los usuarios realicen alguna acción para poder continuar.



Diálogos - Clases

- La clase **Dialog** es la clase de base para los diálogos.
- No crearemos instancias de **Dialog** directamente. Usaremos una de las siguientes subclases:
  - **AlertDialog**: un diálogo que puede mostrar un título, hasta tres botones, una lista de elementos seleccionables o un diseño personalizado.
  - **DatePickerDialog** o **TimePickerDialog**: un diálogo con una IU predefinida que le permite al usuario seleccionar una fecha o una hora

Diálogos - DialogFragment

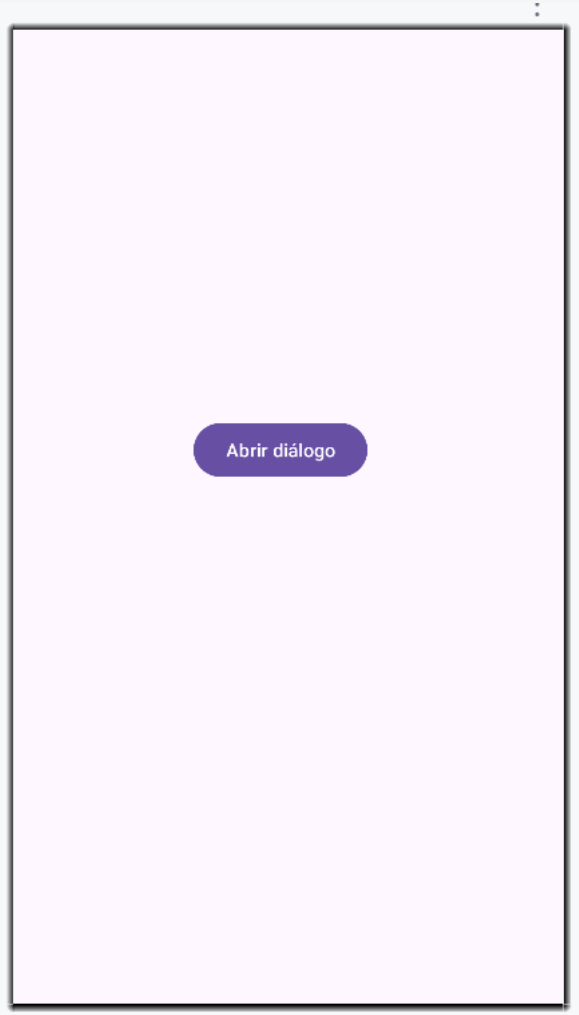
- Aunque se puede usar un diálogo directamente desde un **Activity** la **forma recomendada** es **utilizar** un **DialogFragment** como contenedor para el diálogo.

- El uso de DialogFragment para administrar el diálogo hace que se manejen correctamente los eventos de ciclo de vida, por ejemplo, cuando el usuario presiona el botón Atrás o gira la pantalla.

- Veremos primero como utilizar los diálogos directamente desde un **Activity** y después como integrarlos dentro de un **DialogFragment**.

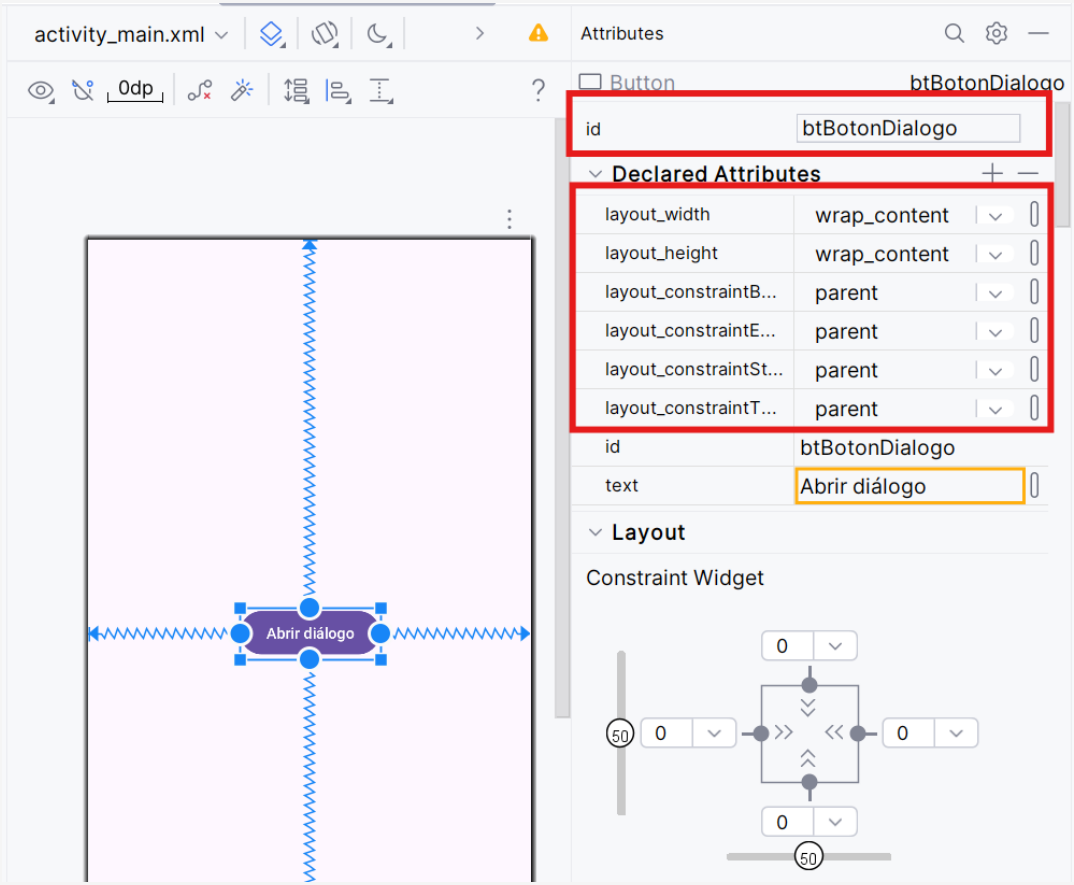
Diálogos - Proyecto

- Crea un proyecto que se llame Dialogos1
- Modifica el Activity principal para que tenga un solo botón con el texto “**Abrir diálogo**”



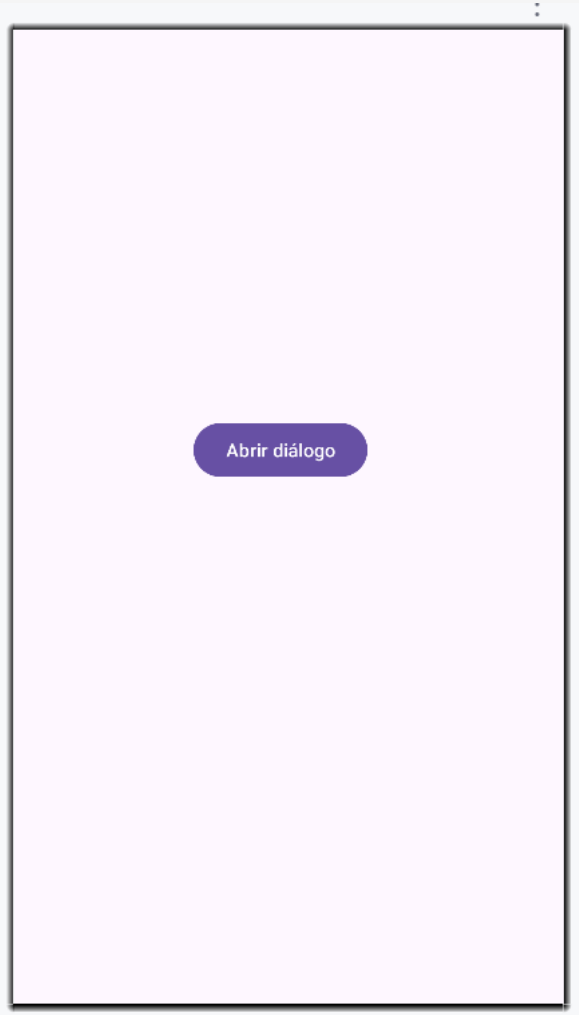
Diálogos - Proyecto

- Crea un proyecto que se llame Dialogos1
- Modifica el Activity principal para que tenga un solo botón con el texto “**Abrir diálogo**”
- Sitúalo en el centro del Activity con constraints
- Asígnale el **id btAbrirDialogo**



Diálogos - Proyecto

- Crea un proyecto que se llame Dialogos1
- Modifica el Activity principal para que tenga un solo botón con el texto "**Abrir diálogo**"
- Sitúalo en el centro del Activity con constraints
- Asígnale el **id btAbrirDialogo**



Diálogos - Proyecto

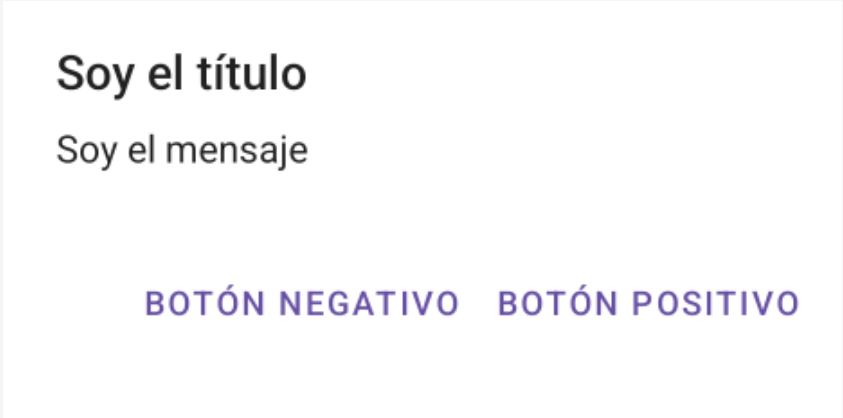
- En el método **onCreate** de MainActivity crea una referencia al botón e **implementa** el método **onClickListener** del **botón**:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_main)
        ViewCompat.setOnApplyWindowInsetsListener( view = findViewById( id = R.id.main)) { v, insets ->
            val systemBars = insets.getInsets( typeMask = WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)
            insets
        }

        var boton: Button =findViewById( id = R.id.btBotonDialogo)

        boton.setOnClickListener {

        }
    }
}
```

Diálogos - Estructura

Hay tres regiones en un diálogo de alerta:

- **Título**: Opcional. Solo se usa cuando el área de contenido está ocupada por un mensaje detallado, una lista o un diseño personalizado.
- **Área de contenido**: Puede mostrar:
  - Un mensaje
  - Una lista
  - Un diseño personalizado.
- **Botones de acción**: puede haber hasta tres botones de acción en un diálogo.



Diálogos – Crear un diálogo de alerta - **AlertDialog**

La clase **AlertDialog.Builder** proporciona APIs que nos permiten crear un **AlertDialog**

1. Creamos un **AlertDialogBuilder**

```
val builder: AlertDialog.Builder = AlertDialog.Builder(this)
```

2. Usando los métodos del **Builder** damos valor a las regiones del diálogo

```
builder.setMessage("Soy el mensaje")
```

```
builder.setTitle("Soy el título")
```

3. Creamos el **AlertDialog**

```
val dialog: AlertDialog = builder.create()
```

4. Mostramos el diálogo

```
dialog.show()
```

Diálogos – Crear un diálogo de alerta - **AlertDialog**

```
boton.setOnClickListener {
    //Creamos un AlertDialog.Builder
    val builder: AlertDialog.Builder = AlertDialog.Builder(this)

    //Damos valor a las zonas del diálogo
    builder.setMessage("Soy el mensaje")
    builder.setTitle("Soy el título")

    //Creamos el AlertDialog
    val dialog: AlertDialog = builder.create()

    //Mostramos el diálogo
    dialog.show()
}
```

Diálogos – Crear un diálogo de alerta - **AlertDialog**

En vez de usar el nombre de la variable AlertDialogBuilder cada vez:

```
    builder.setMessage("Soy el mensaje")
    builder.setTitle("Soy el título")
```

Se suele usar la siguiente nomenclatura en la que solo se pone el nombre de la variable una vez y se van concatenando los métodos set

```
   builder
       .setMessage("Soy el mensaje")
       .setTitle("Soy el título")
```

Diálogos – Crear un diálogo de alerta - **AlertDialog**

```
boton.setOnClickListener {
    //Creamos un AlertDialog.Builder
    val builder: AlertDialog.Builder = AlertDialog.Builder(this)

    //Damos valor a las zonas del diálogo
    builder
        .setMessage("Soy el mensaje")
        .setTitle("Soy el título")

    //Creamos el AlertDialog
    val dialog: AlertDialog = builder.create()

    //Mostramos el diálogo
    dialog.show()
}
```

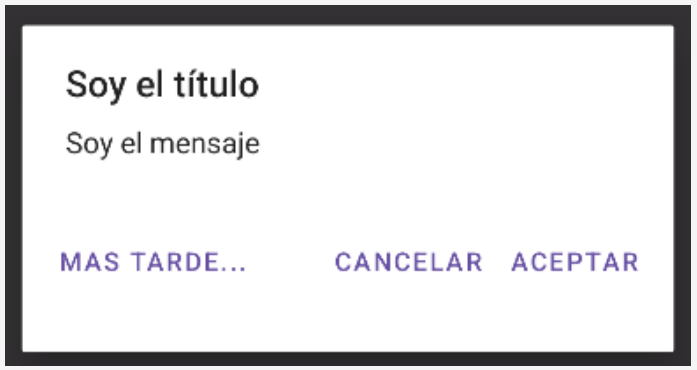
Diálogos – Crear un diálogo de alerta - **Botones**

- Hay tres botones de acción que podemos agregar:
  - **Positivo**: Lo usaremos para aceptar y continuar con la acción (la acción "Aceptar").
  - **Negativo**: Lo usaremos para cancelar la acción.
  - **Neutral**: Usaremos esta opción cuando el usuario no quiera continuar con la acción, pero no necesariamente quiera cancelar. Aparece entre los botones positivo y negativo. Por ejemplo, la acción podría ser "Recordarme más tarde".
- Podemos agregar un solo tipo de botón a un **AlertDialog**. Por ejemplo, **no** podemos tener más de un botón "positivo".

Diálogos – Crear un diálogo de alerta - **Botones**

- Los métodos para incluir los botones son los siguientes:
  - **setPositiveButton**
  - **setNegativeButton**
  - **setNeutralButton**
- Los tres métodos reciben dos parámetros:
  - El **texto** del botón
  - Un **DialogInterface.OnClickListener** que defina la acción que se realizará cuando el usuario presiona el botón.

Diálogos – Crear un diálogo de alerta - **Botones**




```
boton.setOnClickListener {
    //Creamos un AlertDialog.Builder
    val builder: AlertDialog.Builder = AlertDialog.Builder(this)

    //Damos valor a las zonas del diálogo
    builder
        .setMessage("Soy el mensaje")
        .setTitle("Soy el título")
        .setNeutralButton("Mas tarde..."){dialog,wich->
            Toast.makeText(this,"Has pulsado el botón neutro",Toast.LENGTH_SHORT).show()
        }
        .setPositiveButton("Aceptar"){dialog,wich->
            Toast.makeText(this,"Has pulsado el botón positivo",Toast.LENGTH_SHORT).show()
        }
        .setNegativeButton("Cancelar"){dialog,wich->
            Toast.makeText(this,"Has pulsado el botón negativo",Toast.LENGTH_SHORT).show()
        }

    //Creamos el AlertDialog
    val dialog: AlertDialog = builder.create()

    //Mostramos el diálogo
    dialog.show()
}
```

Diálogos – Crear un diálogo de alerta - **Botones**

```
.setPositiveButton("Aceptar"){dialog,wich->
            Toast.makeText(this,"Has pulsado el botón positivo",Toast.LENGTH_SHORT).show()
        }
```

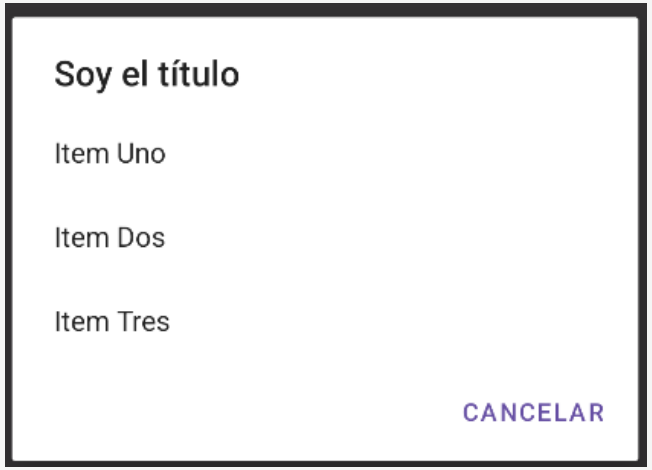
- En el parámetro **dialog** se recibe una referencia al propio diálogo por si se necesita acceder a alguna propiedad o método.
- En el parámetro **wich** se recibe que botón se pulsó para los casos de diálogo que pueda ser necesarios como los ejemplos que veremos a continuación.
- Cuando no se necesita usar estos parámetros es habitual indicarlo poniendo guiones bajos:

```
.setPositiveButton("Aceptar"){_,_->
            Toast.makeText(this,"Has pulsado el botón positivo",Toast.LENGTH_SHORT).show()
        }
```

Diálogos – Crear una lista de opción única

- Para crear una lista de opción única usaremos el método **setItems**().
- En vez de un solo texto para un botón le pasaremos un array con tantos textos como opciones queramos tener.

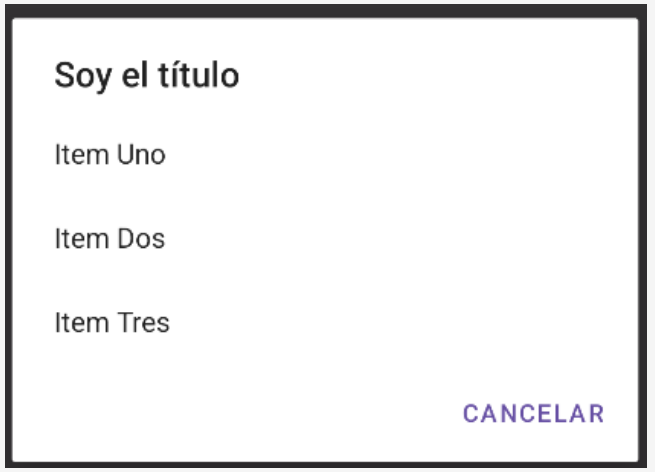
```
builder
    .setTitle("Soy el título")
    .setNegativeButton("Cancelar") { dialog, which ->
        // Añadir acciones
    }
    .setItems(arrayOf("Item Uno", "Item Dos", "Item Tres")) { dialog, which ->
         //Añadir acciones
        }
    }
```



Diálogos – Crear una lista de opción única

- Para crear una lista de opción única usaremos el método **setItems**().
- En vez de un solo texto para un botón le pasaremos un array con tantos textos como opciones queramos tener.
- Al pulsar en una de las opciones el diálogo se cerrará.

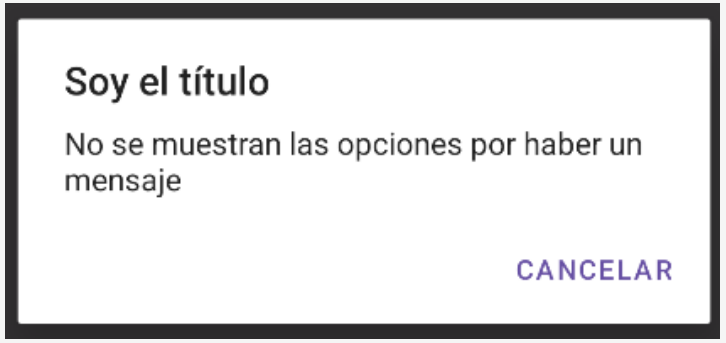
```
builder
    .setTitle("Soy el título")
    .setNegativeButton("Cancelar") { dialog, which ->
        // Añadir acciones    }
    .setItems(arrayOf("Item Uno", "Item Dos", "Item Tres")) { dialog, which ->
         //Añadir acciones
        }
    }
```



Diálogos – Crear una lista de opción única

- Si se añade un mensaje no se mostrarán las opciones:

```
builder
    .setTitle("Soy el título")
    .setMessage("No se muestran las opciones por haber un mensaje")
    .setNegativeButton("Cancelar") { dialog, which ->
        // Añadir acciones     }
    .setItems(arrayOf("Item Uno", "Item Dos", "Item Tres")) { dialog, which ->
       // Añadir acciones
    }
```



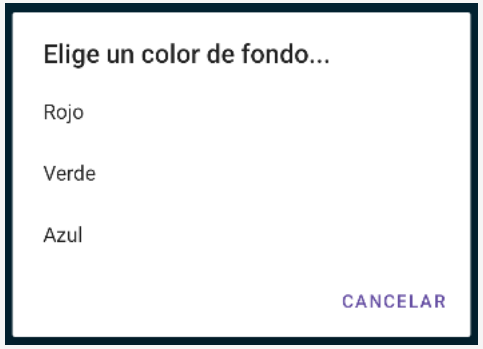
Diálogos – Crear una lista de opción única

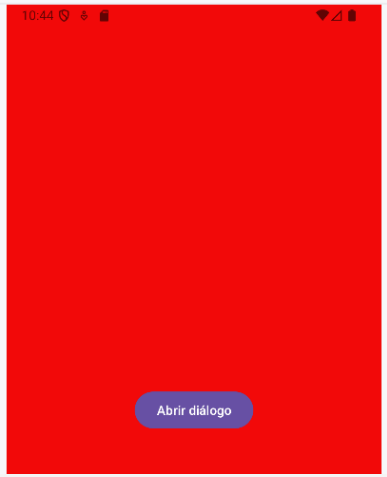
- Para saber que botón se ha pulsado usaremos el parámetro which, que valdrá 0 para el primer elemento, 1 para el segundo etc.

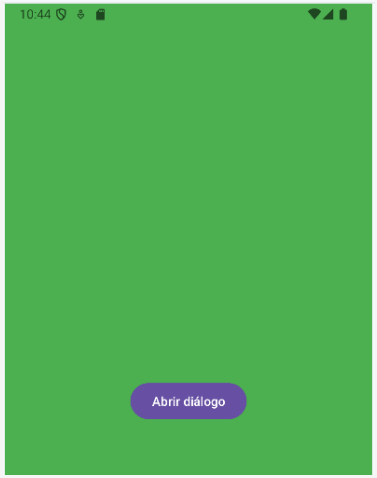
```
builder
    .setTitle("Soy el título")
    .setNegativeButton("Cancelar") { dialog, which ->
        // Añadir acciones
    }
    .setItems(arrayOf("Item Uno", "Item Dos", "Item Tres")) { dialog, which ->
        when(which){
            0->Toast.makeText(this,"Pulsaste en el Item Uno",Toast.LENGTH_SHORT).show()
            1->Toast.makeText(this,"Pulsaste en el Item Dos",Toast.LENGTH_SHORT).show()
            2->Toast.makeText(this,"Pulsaste en el Item Tres",Toast.LENGTH_SHORT).show()
        }
    }
```

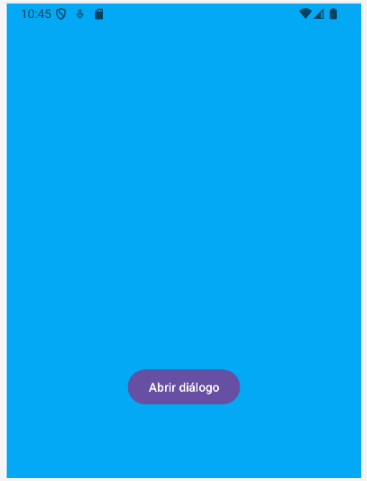
Diálogos – Ejercicio lista de opción única

- Crea un diálogo de elección de una sola opción que permita elegir entre 3 colores y que ponga el color de fondo del activity del color elegido.







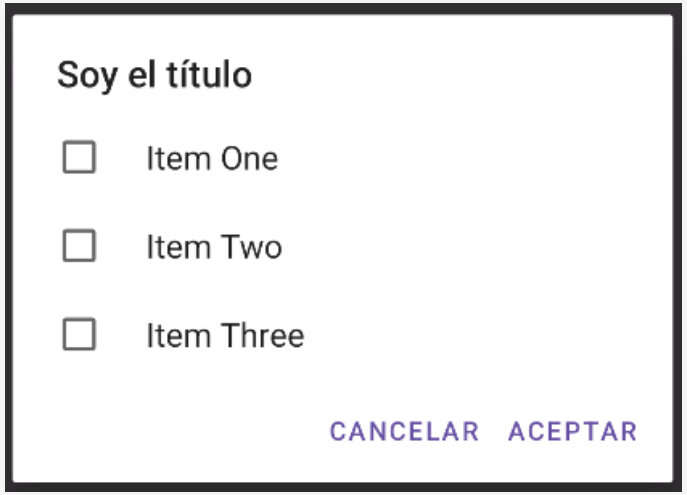


- Establece 3 constantes de color
- Para dar valor al fondo obtén una referencia al ConstraintLaout y usa el método setBackgroundColor
  - `esteLayout.setBackgroundColor(getResources().getColor(R.color.rojo,theme))`

Diálogos –Lista de botones de selección múltiple

- Para crear una lista de selección múltiple usaremos el método **setMultiChoiceItems()**
- Igual que en la lista de opción única le pasaremos un array con tantos textos como opciones queramos tener.

```
//Creamos un array de opciones
var opciones = arrayOf("Item One", "Item Two", "Item Three")

//Damos valor a las zonas del diálogo
builder
    .setTitle("Soy el título")
    .setPositiveButton("Aceptar") { dialog, which ->
        // Añadir acciones
    }
    .setNegativeButton("Cancelar") { dialog, which ->
        // Añadir acciones
    }
    .setMultiChoiceItems(
        opciones, null) { dialog, which, isChecked ->
        //Añadir acciones
    }
```



Diálogos –Lista de botones de selección múltiple

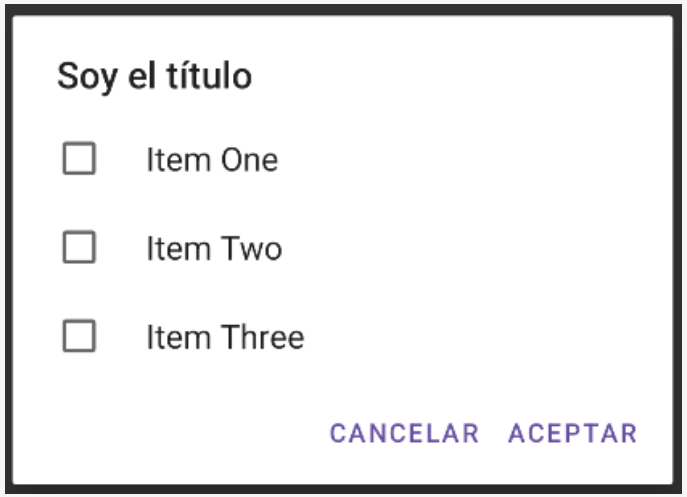
- El manejador de click de **setMultiChoiceItems** tiene 3 parámetros:
- Los dos primeros tienen la misma función que en los diálogos vistos anteriormente.
- El tercero (**which**) es un valor booleano que indica si el botón que se pulso quedó **marcado** (**true**) o **desmarcado** (**false**)

```
builder
    .setTitle("Soy el título")
    .setPositiveButton("Aceptar") { dialog, which ->
        // Añadir acciones
    }
    .setNegativeButton("Cancelar") { dialog, which ->
        // Añadir acciones
    }
    .setMultiChoiceItems(
        opciones, null) { dialog, which, isChecked ->
        val pulsado=opciones[which] //Obtenemos el nombre del item pulsado
        var estado= " que ha quedado desmarcado"
        if(isChecked) estado="que ha quedado marcado"
        Toast.makeText(this,"Se ha pulsado $pulsado $estado",Toast.LENGTH_SHORT).show()

    }
```

Diálogos –Lista de botones de selección múltiple

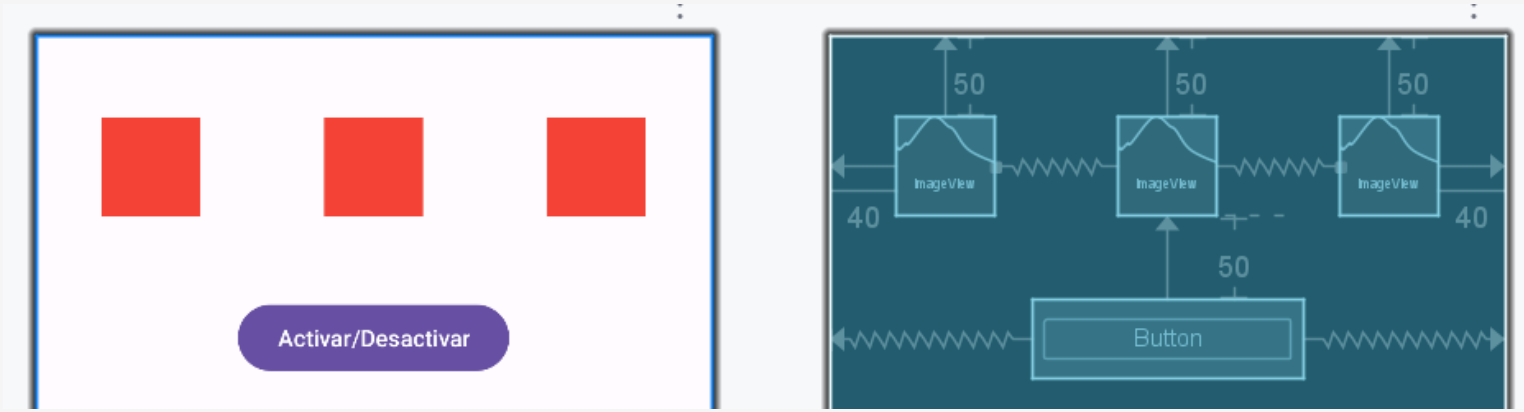
- Para saber que ítems se seleccionaron y establecer los que están marcados por defecto se usará un array de booleanos que se pasará como segundo parámetro al método setMultiChoiceItems

```
//Creamos un array para saber los items seleccionados
var itemSeleccionados = booleanArrayOf(true,false,false)

//Damos valor a las zonas del diálogo
builder
    .setTitle("Soy el título")
    .setPositiveButton("Aceptar") { dialog, which ->
        // Añadir acciones
        //Toast.makeText(this,"$cadena",Toast.LENGTH_LONG).show()
    }
    .setNegativeButton("Cancelar") { dialog, which ->
        // Añadir acciones
    }
    .setMultiChoiceItems(
        opciones, itemSeleccionados) { dialog, which, isChecked ->
        //Añadir acciones
        var cadena=""

        for(i in itemSeleccionados.indices){
            if(itemSeleccionados[i])
                cadena = cadena + opciones[i] + " seleccionado "
        }
        Toast.makeText(this,"$cadena",Toast.LENGTH_LONG).show()
    }
```



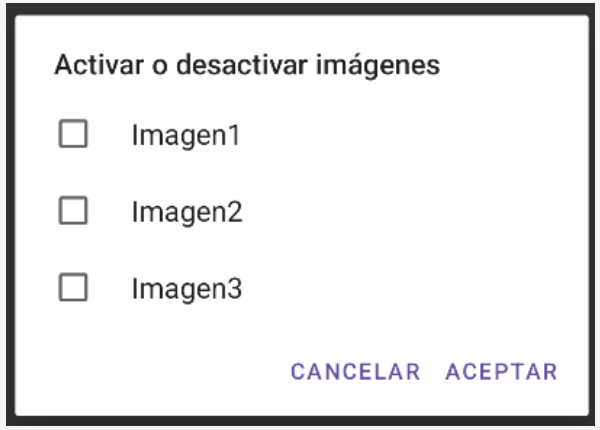
Diálogos –Ejercicio Lista de botones de selección múltiple

- Crea un proyecto Nuevo que se llame EjercicioListaSeleccionMultiple
- Crea el siguiente diseño que tiene 3 ImageViews y un botón. Los imageViews no tienen una imagen asociada simplemente un color de fondo rojo (define el color en las recursos de constantes de color res->values->colors.xml).
- Aunque hemos visto que no es lo más recomendable, para este ejercicio asigna a los imageViews un ancho y alto específico de 60dp.



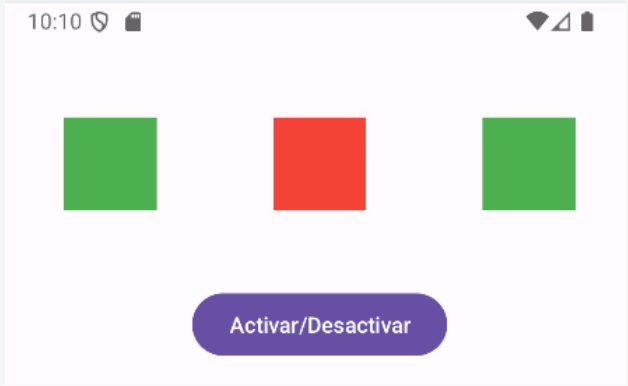
Diálogos –Ejercicio Lista de botones de selección múltiple

- Al pulsar el botón se mostrará el siguiente diálogo.



- Por defecto todas las imágenes están desactivadas (se muestran con color rojo).
- En el cuadro de diálogo podemos marcar y desmarcar imágenes. Al pulsar en el botón Aceptar tendremos que actualizar los ImageViews de forma que las imágenes que hayamos dejado marcadas pasen a tener color Verde y las desmarcadas color Rojo.

Diálogos –Ejercicio Lista de botones de selección múltiple



Orientaciones:

- Crea un atributo en la clase MainActivity para almacenar el array de estados de las casillas. De esta forma siempre tendremos disponible el estado de activación/desactivación.

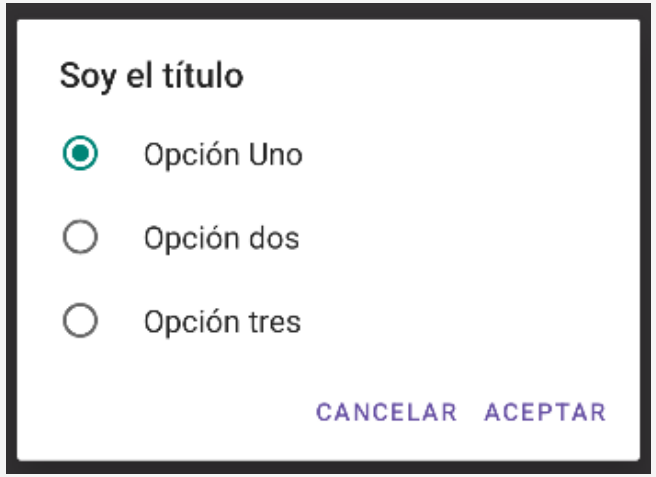
Diálogos – Crear una lista de botones de selección única

Para agregar una lista de botones de selección única usaremos el méotod **setSingleChoiceItems()** que recibe dos parámetros:

- El primero es una lista con el texto de las opciones. **Ejemplo**: *arrayOf*("Opción Uno", "Opción dos", "Opción tres")
- El segundo es un entero que indica cual de los botones está marcado por defecto. Siendo 0 el primero, 1 el segundo etc.

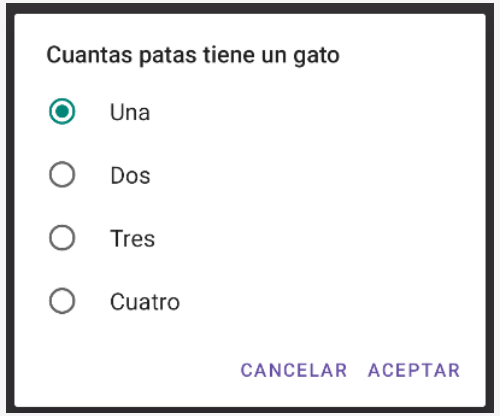
Diálogos – Crear una lista de botones de selección única

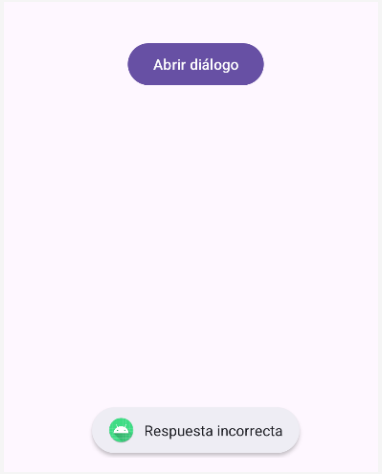
```
//Creamos un AlertDialog.Builder
val builder: AlertDialog.Builder = AlertDialog.Builder(this)
//Creamos un array con las opciones
var opciones = arrayOf("Opción Uno", "Opción dos", "Opción tres")
var cadena = ""
builder
    .setTitle("Soy el título")
    .setPositiveButton("Aceptar") { dialog, which ->
        Toast.makeText(this, cadena, Toast.LENGTH_LONG).show()
    }
    .setNegativeButton("Cancelar") { dialog, which ->
        // Do something else.
    }
    .setSingleChoiceItems(
        opciones, 0
    ) { dialog, which ->
        cadena = "El elemento seleccionado es el " + which
    }
```

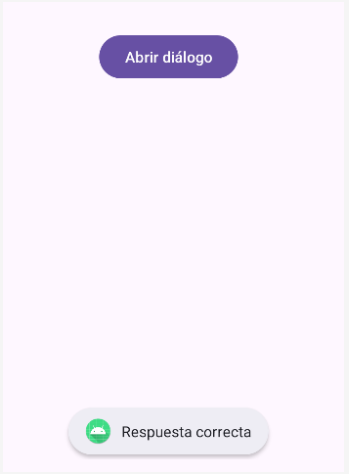


Diálogos – Ejercicio de lista de selección única

- Crea un diálogo de botones de selección de elección única que presente una pregunta y que compruebe si la respuesta es correcta.



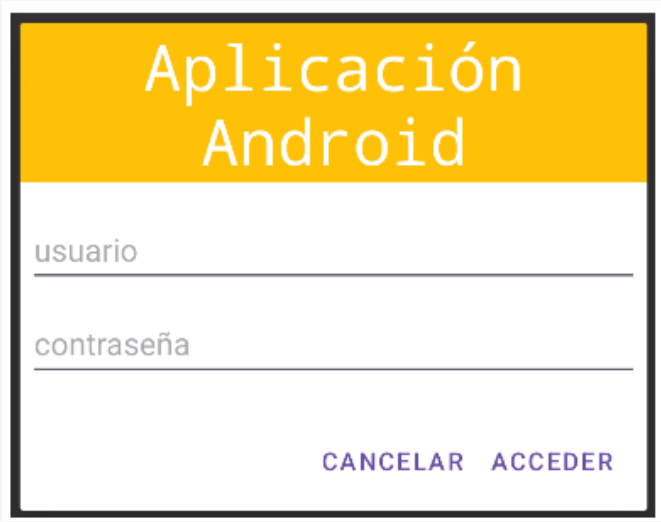




- Crea una clase Pregunta con el enunciado, respuestas y el número de respuesta correcta.
- Al método que cree el diálogo pásale como parámetro una pregunta.

Diálogos – Personalizar el diálogo

Si queremos un diseño personalizado en un diálogo, crearemos un diseño y lo agregaremos a un **AlertDialog** llamando a **setView()** en el objeto **AlertDialog.Builder.**



Diálogos – Personalizar el diálogo

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

    <TextView
        android:id="@+id/cabecera"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#FFC107"
        android:fontFamily="monospace"
        android:text="Aplicación Android"
        android:textColor="@color/white"
        android:gravity="center"
        android:textSize="36dp" />

    <EditText
        android:id="@+id/usuario"
        android:inputType="textEmailAddress"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:layout_marginLeft="4dp"
        android:layout_marginRight="4dp"
        android:layout_marginBottom="4dp"
        android:hint="usuario" />
    <EditText
        android:id="@+id/contraseña"
        android:inputType="textPassword"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="4dp"
        android:layout_marginLeft="4dp"
        android:layout_marginRight="4dp"
        android:layout_marginBottom="16dp"
        android:fontFamily="sans-serif"
        android:hint="contraseña"/>

</LinearLayout>
```

Diálogos – Personalizar el diálogo

Para aumentar el diseño de nuestro Diálogo, obtendremos un **LayoutInflater** con **getLayoutInflater()** y llamaremos a **inflate()**. El primer parámetro de **inflate()** es el ID del recurso de diseño y el segundo es una vista superior para el diseño.

```
// Obtenemos el inflater.
val inflater = layoutInflater;

//Cargamos el diseño y lo almacenamos en una variable
val dialogView = inflater.inflate(R.layout.dialogo_login, null)
```

Diálogos – Personalizar el diálogo

- Asignamos el diseño al diálogo con el método **setView()**
- Mediante la variable en la que hemos cargado el diseño podemos obtener referencias a los elementos del mismo con **findViewById**

```
// Cargamos el diseño y se lo asignamos al diálogo.
builder.setView(dialogView)
    // Añadimos los botones
    .setPositiveButton("Acceder") { dialog, id ->
        // Código para validar al usuario
        val etNombre: EditText=dialogView.findViewById(R.id.usuario)
        val etContra: EditText=dialogView.findViewById(R.id.contraseña)
        if(etNombre.text.toString()==("user") && etContra.text.toString()==("1234"))
            Toast.makeText(this, "Login correcto", Toast.LENGTH_LONG).show()
        else
            Toast.makeText(this, "Login incorrecto", Toast.LENGTH_LONG).show()
        }
    .setNegativeButton("Cancelar") { dialog, id ->
            //Hacer algo
    }
```

Diálogos – Integrar el dialogo en un DialogFragment

- Para integrar un diálogo en un fragment crearemos una clase que extienda **DialogFragment.**
- En el método de devolución de llamada **onCreateDialog()** es donde crearemos el diálogo con cualquiera de los formatos que vimos en los apartados anteriores.

```
class StartGameDialogFragment : DialogFragment() {
    override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
        return activity?.let {
            val builder = AlertDialog.Builder(it)
            builder.setMessage("Comenzar el juego")
                .setPositiveButton("Empezar") { dialog, id ->
                    // Código para empezar el juego
                }
                .setNegativeButton("Cancelar") { dialog, id ->
                    // Código cuando el usuario cancela el diálogo
                }
            // Creamos el AlertDialog y lo retornamos
            builder.create()
        } ?: throw IllegalStateException("Activity cannot be null")
    }
}
```

Diálogos – Integrar el dialogo en un DialogFragment

Para crear un diálogo desde un Activity por ejemplo MainActivity crearemos una instancia de la clase anterior y llamaremos al método show():

```
StartGameDialogFragment().show(supportFragmentManager, "GAME_DIALOG")
```

- **supportFragmentManager**: Este es el gestor (manager) responsable de manejar los *fragments* dentro de tu actividad (generalmente se obtiene a través de supportFragmentManager en una FragmentActivity o AppCompatActivity). Se encarga de gestionar el ciclo de vida del *fragment* y de añadirlo a la jerarquía de vistas de la actividad.

- **"GAME_DIALOG"**: Esta es una **etiqueta** (*tag*) (un identificador de cadena único) asignada al *fragment* de diálogo. Es útil porque más tarde podemos recuperar la misma instancia del diálogo usando esta etiqueta, por ejemplo:

```
val dialog = supportFragmentManager.findFragmentByTag("GAME_DIALOG")
```