Módulo: Programación Multimedia y dispositivos móviles.
Unidad de Trabajo Nº1: Introducción al desarrollo de aplicaciones móviles.
Actividad de Clase 4: Controles RadioGroup, RadioButton, CheckBox y Spinner y ViewBinding.

1. **Objetivo general.**

   Manejar diferentes componentes (RadioGroup, RadioButton, Checkbox y Spinner) en Android Studio.

2. **Metodología.**

   Cada alumno realizará la app.

3. **Descripción.**

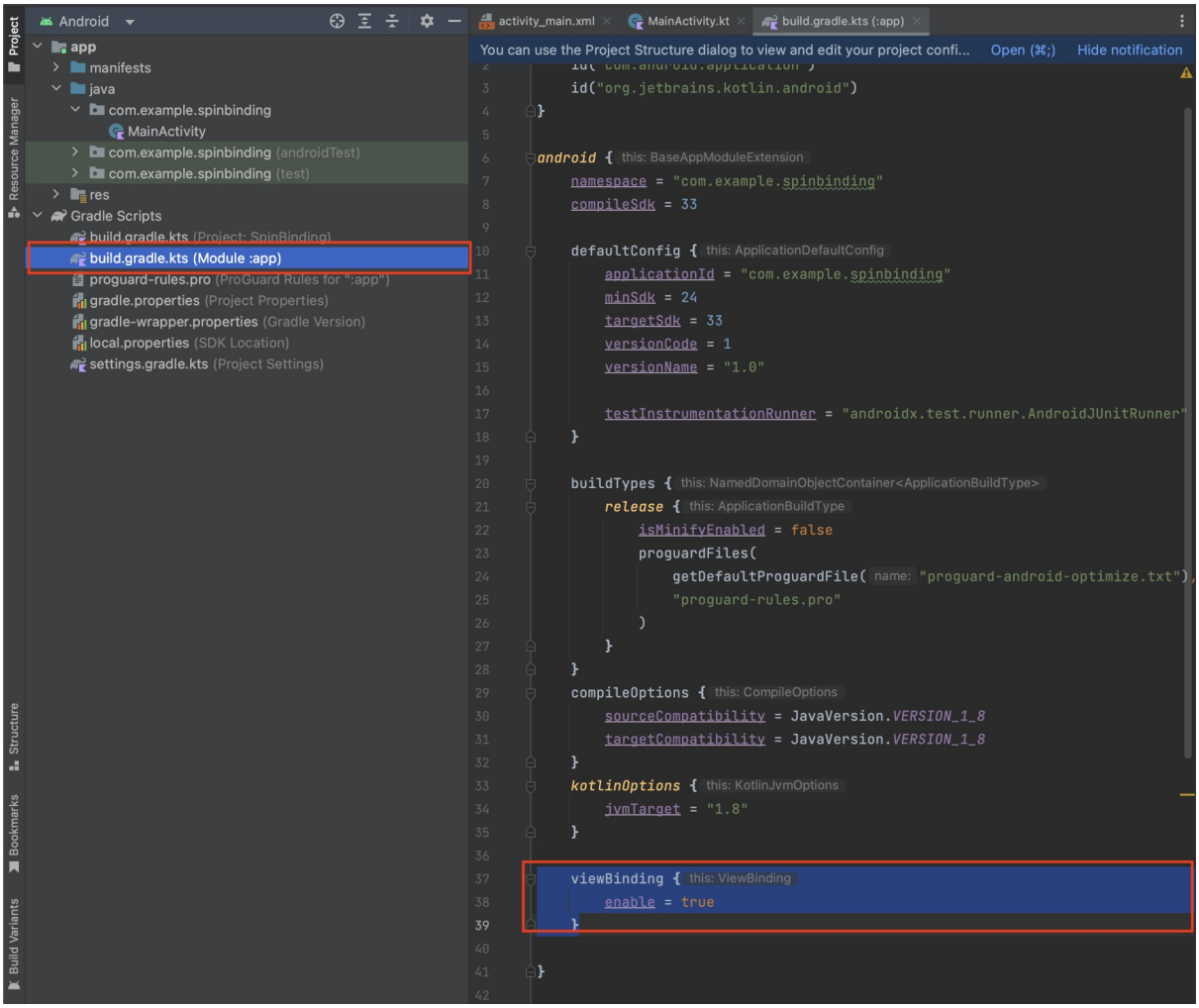
   Realiza lo siguiente:

Primera Parte: Utilizar View Binding en vez findViewById

1. Para vincular las vistas (los componentes) hemos utilizado hasta ahora findViewById . Como en el siguiente ejemplo donde accedemos a un componente definido, en este caso un spinner que utilizaremos más adelante en la actividad.

```
//Accedemos al spinner
val spin = findViewById<Spinner>(R.id.spinner2)
```

2. Como hemos visto este código funciona. Pero tendremos que hacer lo mismo con cada componente que queramos usar generando un montón de código repetido. Por ello, tenemos otra forma de hacerlo y es usando **view binding**. Para configurarlo debemos ir a **build.gradle.kts (Module :app)** y allí buscaremos la sección android {} y en esta sección añadiremos lo siguiente:



3. Sincronizada el Gradle.

Segunda Parte: Controles RadioGroup, RadioButton y CheckBox.

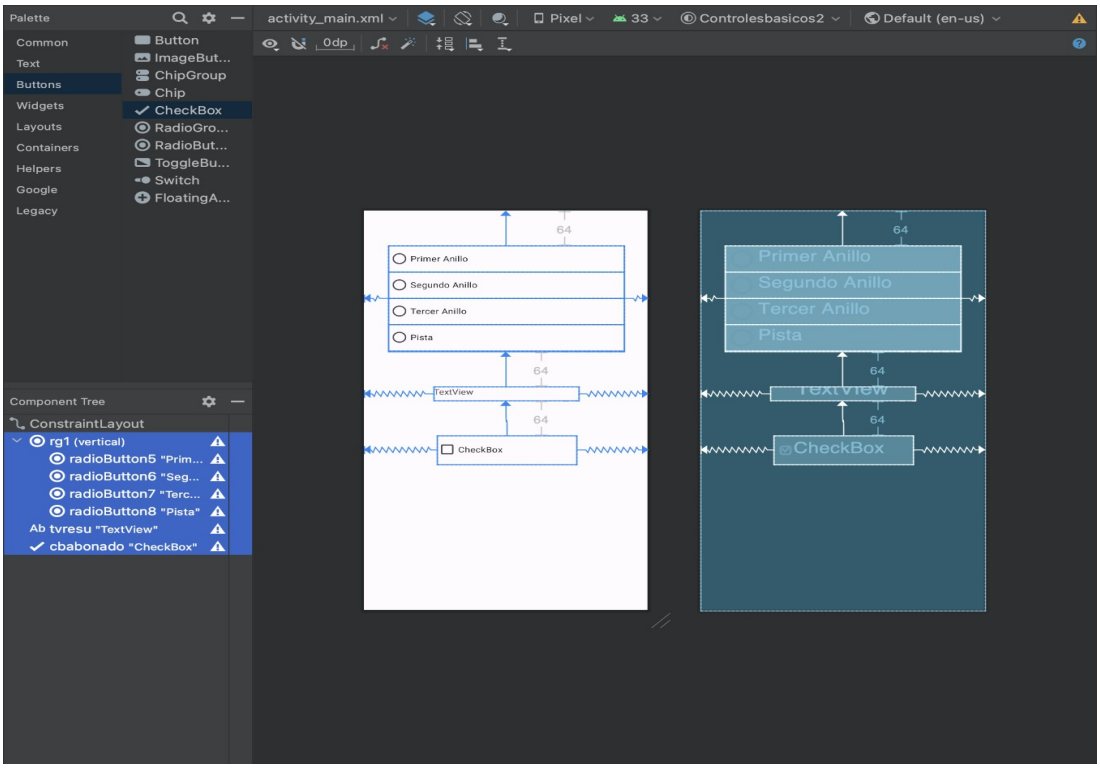
4. Vamos a crear una app que va a elegir la zona del Gran Canaria Arena. Las zonas son las siguientes:
   - Primer Anillo, Segundo Anillo, Tercer Anillo, Pista
   - El usuario sólo podrá elegir una zona.
   - Información al usuario que sólo puede seleccionar una zona.
   - Marcar si el usuario es abonando o no.

5. Puedes añadir los controles de forma gráfica, arrastrando los Radiobutton dentro del Radiogroup, o bien, en modo texto en el XML (prueba a hacerlo de las dos maneras).

6. La Activity contendrá los siguientes controles:
7. Un Radiogroup y dentro un Radiobutton para cada zona
8. Un TextView para mostrar información al usuario que sólo puede seleccinar una zona
9. Un Checkbox para que el usuario seleccione si es o no abonado.

Nota: Las imágenes de referencia que aparecen en el documento no son las solicitadas en el programa solicitado.

10. La Activity quedaría de forma similar a la que aparece en la siguiente imagen donde vemos el RadioGroup, los RadioButton, TextView y Checkbox en modo diseño: En las imágenes tiene los nombre por defectos (realizado antes de modificar las propiedades **id** y **text**



Y lo mismo en código (fichero xml).

```xml
<?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">
    <RadioGroup
        android:id="@+id/rg1"
        android:layout_width="341dp"
        android:layout_height="194dp"
        android:layout_marginTop="64dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">
        <RadioButton
            android:id="@+id/radioButton5"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Primer Anillo" />
        <RadioButton
            android:id="@+id/radioButton6"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Segundo Anillo" />
        <RadioButton
            android:id="@+id/radioButton7"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Tercer Anillo" />
        <RadioButton
            android:id="@+id/radioButton8"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Pista" />
    </RadioGroup>

    <TextView
        android:id="@+id/tvresu"
        android:layout_width="209dp"
        android:layout_height="25dp"
        android:layout_marginTop="64dp"
        android:text="TextView"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/rg1" />

    <CheckBox
        android:id="@+id/cbabonado"
        android:layout_width="202dp"
        android:layout_height="53dp"
        android:layout_marginTop="64dp"
        android:text="CheckBox"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/tvresu" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

11. Una vez finalizado el diseño de la Activity donde hemos trabajado con el archivo xml (activity_main.xml) donde se definen los controles visuales de la ventana que estamos creando. Abrimos el archivo MainActivity.kt.

12. A continuación, vamos a ver el código de la clase **MainActivity** pero viendo las diferentes partes. Vamos a empezar primero por el manejo del **View Binding**. una vez activado el view binding, cada activity que creemos, creará por detrás el binding, es decir, si nosotros tenemos **MainActivity** que tiene un layout llamado activity_main.xml, automáticamente se creará **ActivityMainBinding**. Lo primero es crear una variable de clase que se del tipo **ActivityMainBinding** pero como dicha variable la vamos a **inicializar** más tarde (en el método **onCreate()**) la vamos a declarar del tipo **lateinit** este tipo en **Kotlin** nos permite declararla e inicializarla posteriormente.

```
class MainActivity : AppCompatActivity() {

    // Declaramos la variable del tipo ActivityMainBinding. La podemos inicializar más tarde porque es lateinit
    private lateinit var binding: ActivityMainBinding
```

13. Ahora inicializaremos la variable, en el método onCreate() de nuestra activity. Después solo tendremos que cambiar la función setContentView (que ahora mismo mandamos el layout) y le ponemos la ruta de nuestro binding.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    // Inicializamos la variable creandolo y vinculandolo
    binding = ActivityMainBinding.inflate(layoutInflater)
    // Mostramos la vista raiz.
    setContentView(binding.root)
```

14. Ahora ya podríamos acceder a nuestros componentes simplemente usando **binding.LaIdDelComponente**. Como podemos observar a continuación donde accedemos a los componentes sin tener que utilizar el **findViewById**.

15. Para poder acceder a los objetos visuales definidos en el archivo activity_main.xml (El radioGroup (radiog1), el TextView (tv) y el checkbox (cb)). .

```
//Accedemos a los componentes utilizando binding
val radiog1 = binding.rg1
val tv = binding.tv1
val cb = binding.cb1
```

16. Ahora vemos como creamos el listener para el RadioGroup. En Kotlin en vez de utilizar switch case. Utilizamos when que es similar.

```
radiog1.setOnCheckedChangeListener { group, checkedId ->
    when(checkedId) {
        binding.rb1.id -> {
            tv.text = binding.rb1.text
        }

        binding.rb2.id -> {
            tv.text = binding.rb2.text
        }

        binding.rb3.id -> {
            tv.text = binding.rb3.text
        }
    }
}
```

17. Y ahora vemos el listener para el CheckBox

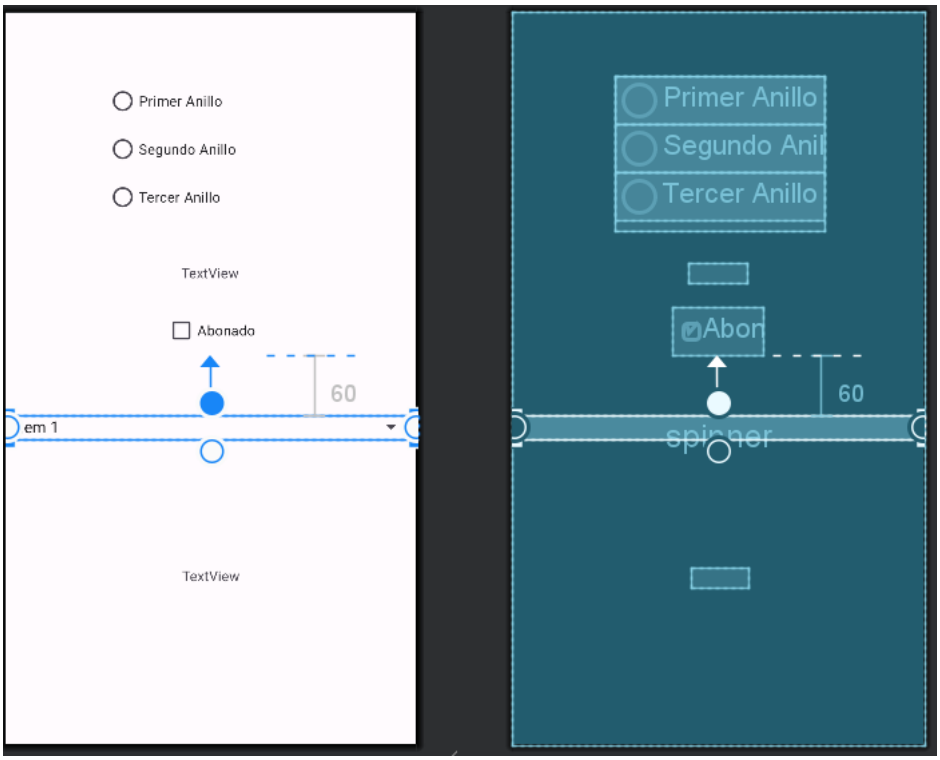
```
cb.setOnCheckedChangeListener { buttonView, isChecked ->
    var mensaje =""
    if (isChecked) {
        mensaje = "Abonado"
    } else {
        mensaje = "No Abonado"
    }
   Toast.makeText( context: this, mensaje, Toast.LENGTH_LONG).show()
}
```

18. Una vez creados los listener para ambos controles. Ejecuta la aplicación. Y comprueba su funcionamiento.

Tercera Parte: Añadir un spinner.

En esta tercera parte vamos a añadir un **spinner** (que incluya una lista por ejemplo, Provincias, Islas, Series, películas, juegos, etc), un **TextView** para mostrar el item del spinner seleccionado

19. Añade los diferentes componentes **Spinner** y (**TextView**),, en la siguiente imagen puedes ver un ejemplo..



20. Ahora accedemos al spinner y el TextView añadidos usando **binding.LaIdDelComponente**

```
val espin = binding.spinner
val tvre = binding.tvresu
```

21. **Spinner**: es un componente permite seleccionar un valor de entre un conjunto, lo que conocemos como ComboBox. Este grupo de valores puede almacenarse en un array o bien, en un cursor si lo cargamos desde una base de datos. También tenemos la posibilidad de definir el array en el fichero **strings.xml**, con diferentes opciones puedes elegir el listado que quieras en el ejemplo aparecen las provincias de Galicia

(pero no tiene que ser el que utilices). En la siguiente imagen puedes ver el contenido del fichero **strings.xml** (el tuyo no debe ser igual)

```xml
<resources>
    <string name="app_name">Viewbinding24</string>
    <string-array name="provincias">
        <item>A Coruña</item>
        <item>Lugo</item>
        <item>Orense</item>
        <item>Pontevedra</item>
    </string-array>
</resources>
```

22. Ahora vamos a crear el adaptador para vincular nuestros datos con el Spinner. Para ello utlizando el ***createFromResource()*** para hacer referencia al array XML que que creamos en strings.xml y luego asignamos el adaptador al spinner.

```
val pro = ArrayAdapter.createFromResource( context: this, R.array.provincias, com.google.android.material.R.layout.support_simple_spinner_dropdown_item)
espin.adapter = pro
```

Otra opción es crear el array en Kotlin el código sería el siguiente:

```
val islas = arrayOf("La Graciosa", "Lanzarote", "Fuerteventura", "Gran Canaria")
val adap = ArrayAdapter( context: this, androidx.appcompat.R.layout.support_simple_spinner_dropdown_item, islas)
espin.adapter = adap
```

23. Ahora vamos a ver los eventos lanzados por el Spinner, el más utilizado será el generado al seleccionarse una opción de la lista desplegable, **onItemSelected**. Para capturar este evento se procederá como lo hemos visto para otros controles anteriormente, asignándole su controlador mediante la propiedad **onItemSelectedListener.** A diferencia de los controles anteriores, para este evento definimos dos métodos, el primero de ellos (**onItemSelected**) que será llamado cada vez que se selecciones una opción en la lista desplegable, y el segundo (**onNothingSelected**) que se llamará cuando no haya ninguna opción seleccionada (esto puede ocurrir por ejemplo si el adaptador no tiene datos). Además, vemos cómo para recuperar el dato seleccionado utilizamos el método **getItemAtPosition()** del

parámetro **AdapterView (parent en el ejemplo)** que recibimos en el evento. En este ejemplo mostramos el valor seleccionado en el TextView para el mensaje.

```
espin.onItemSelectedListener = object : AdapterView.OnItemSelectedListener {
    override fun onItemSelected(
        parent: AdapterView<*>?,
        view: View?,
        position: Int,
        id: Long
    ) {
        tvre.text = "Has seleccionado ${parent!!.getItemAtPosition(position)}"
    }

    override fun onNothingSelected(parent: AdapterView<*>?) {
        tvre.text = "No has seleccionado nada"
    }
}
```

24. Ejecuta el programa y observa su funcionamiento..

25. A continuación, puedes ver la aplicación de ejemplo en funcionamiento en el siguiente video:

https://drive.google.com/file/d/1TJDL3L05PvKscZO_l77_4piVMoxnRkLT/view?usp=sharing