Botones

Un botón es un control con texto o imagen que realiza una acción cuando el usuario lo presiona. En la paleta de diseño, podemos ver los siguientes tipos de botones:

- **Button**: Un botón es un control con texto o imagen que realiza una acción cuando el usuario lo presiona. La clase que lo represente es Button que hereda de TextView, y puedes referirte a él dentro de un layout con la etiqueta <Button>.
  El texto que especifica su acción indícalo en el atributo **android:text**.
  El color de fondo de los botones se puede cambiar con el atributo **android.backgroundTint**.

- **ImageButton** funciona exactamente igual que Button, solo que, en lugar de mostrar un texto en su background, viene una imagen.

  Para cambiar la imagen de un image button usa el atributo **android:src**. Obviamente su valor es un drawable.

Los atributos más comunes de los botones son los que se muestran a continuación:

Atributo	Descripción
`android:text`	Permite cambiar el texto de un botón
`android:background`	Se usa para cambiar el fondo del botón. Puedes usar un recurso del archivo `colors.xml` o un *drawable*.
`android:enabled`	Determinar si el botón está habilitado ante los eventos del usuario. Usa `true` (valor por defecto) para habilitarlo y `false` en caso contrario.
`android:id`	Representa al identificador del botón para diferenciar su existencia de otros views.
`android:onClick`	Almacena la referencia de un método que se ejecutará al momento de presionar el botón.
`android:textColor`	Determina el color del texto en el botón
`android:drawable*`	Determina un drawable que será dibujado en la orientación establecida. *Por ejemplo*: Si usas el atributo `android:drawableBottom`, el drawable será dibujado debajo del texto.

Es de esperar que un botón dispara un evento cuando el usuario hace click. Para procesar el evento existen varias formas de hacerlo:

- Usar el atributo **android:onClick**: Para ello se requiere que el método cumpla con las siguientes condiciones:
  - Que sea público,
  - Que no devuelva nada (que sea tipo void)
  - Que reciba un parámetro del tipo View
  - Debe declararse en la actividad que usa el mismo layout

Ejemplo: Validar el número de teléfono de un Text View al presionar un Button.

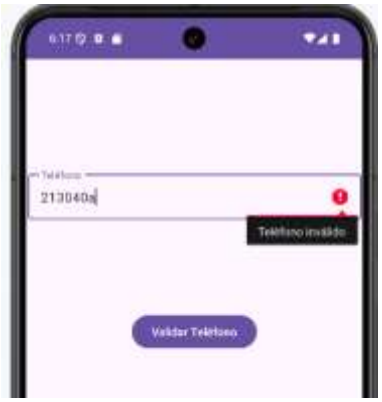
En la Activity crear la siguiente función:

```
fun validarTelefono(view: View){
    var telefono = binding.telefono.text.toString()
    if (Patterns.PHONE.matcher(telefono).matches()) {
        binding.telefono.setError(null)
            }
    else {
        binding.telefono.setError("Teléfono inválido")
    }
}
```

En el atributo android:onClick asignar el nombre de la función validarTelefono. Si la función está bien creada, el motor de Android Studio debe sugerirte la función:

```
android:onClick="validarTelefono"
```

Comprobamos que funciona correctamente:



— Usar Escucha Anónima **OnClickListener:** Otra forma es crear una instancia anónima de la interfaz View.OnClickListener para manejar los eventos del botón. Esto requiere usar el método setOnClickListener() y eliminar el atributo android.onClick:

```
class MainActivity : AppCompatActivity() {

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

        binding.button.setOnClickListener(){
            validarTelefono(binding.telefono.text.toString())
        }
    }

    fun validarTelefono(telefono: String){
        if (Patterns.PHONE.matcher(telefono).matches()) {
            binding.telefono.setError(null)
                }
        else {
            binding.telefono.setError("Teléfono inválido")
        }
    }
```

Ejemplo: Este caso puede ser útil para iniciar otra actividad al presionar un botón. Lo primero que haremos será crea una nueva actividad desde **File > New > Activity > Empty Views Activity,** a la que llamaremos ActividadNueva.

Lo primero que haremos siempre, será crear el binding en ActividadNueva:

```
class ActividadNueva : AppCompatActivity() {
    private lateinit var binding: ActivityActividadNuevaBinding

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

        binding = ActivityActividadNuevaBinding.inflate(layoutInflater)
        setContentView(binding.root)
    }
}
```

Lo siguiente es ir a MainActivity y obtener la instancia del botón que tenemos, luego invoca el método setOnClickListener() desde la intancia:

```
class MainActivity : AppCompatActivity() {

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

        binding.button.setOnClickListener {
            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 de inicio, 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().

```
class MainActivity : AppCompatActivity() {

    companion object {
        val EXTRA_TELEFONO: String = "TELEFONO"
        val EXTRA_MENSAJE: String = "MENSAJE"
    }

    private lateinit var binding: ActivityMainBinding

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

        binding.button.setOnClickListener {

            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().

```
class ActividadNueva : AppCompatActivity() {
    private lateinit var binding: ActivityActividadNuevaBinding

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

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

        //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)
        binding.textView.setText("$mensaje - $telefono")

    }
}
```

Ejercicio 1: Realizar un programa que contenga dos Activity. En el primero que solicite el ingreso de una clave (control Password) Si ingresa la clave "abc123" activar el segundo Activity mostrando en un TextView un mensaje de bienvenida seguido de la contraseña introducida en el primer activity. Mostrar en Toast si se ingresa la clave incorrecta en el primer Activity.

Ejercicio2: Crea una activity funcional como la siguiente. Las cajas editables solo deben admitir números.

Existen cuatro botones con las cuatro operaciones básicas.

Al pulsar un botón saldrá un mensaje largo indicando la operación que se está realizando abajo (Toast) y el resultado de la operación en la caja "Resultado".

Si la operación no está permitida, como ocurre con la división por cero, indicarlo con un Toast.

Investiga cómo poner un botón que reinicie la aplicación (botón limpiar).

Ejercicio3: Realiza una calculadora sencilla que tenga una funcionalidad parecida a esta pantalla que ya tenías diseñada.

Realiza todo el control de errores con mensajes temporales (tipo Toast).

- **RadioButton**: Un radiobutton es un control de selección que proporciona la interfaz de Android para permitir a los usuarios seleccionar una opción de un conjunto. Un radiobutton hereda de Button, por lo que tiene sus mismas propiedades y la forma de manejar eventos es igual a la vista con Button.

  Son ideales para elegir uno de varios elementos con exclusión mutua, es decir, la selección de un radio button obliga a descartar la de otro, permitiendo solo a un ítem estar activo.

  La exclusión la logras con el componente **RadioGroup**, el cual agrupa los radio buttons para permitir la selección de uno solamente. En radio group usa la propiedad layout_height y layout_width con el valor wrap_content para que se ajuste al tamaño de los radio buttons que contiene.

  Un Radio Button puede atravesar por 5 estados al igual que los otros controles de selección.
  - **hover**: el radio button está a la espera de eventos (el ratón está sobre el botón)
  - **focused**: el sistema de navegación de android enfocó al radio button
  - **pressed**: el usuario mantiene presionado el radio button
  - **disabled**: el radio button no está habilitado para interacción
  - **disabled-focused**: combinación de **disabled** + **focused** (El botón está deshabilitado pero el sistema de navegación lo enfoca)

  Un dato importante. RadioGroup extiende de LinearLayout, por lo que es posible usar el atributo **android:orientation** para la alineación de los radio buttons (horizontal / vertical).

  Usa el atributo **android:checked** para determinar el estado de cada radio button en tiempo de diseño (true / false) si quieres que uno aparezca seleccionado por defecto. Otra alternativa para determinar el radio button marcado por defecto es a través de la propiedad **checkedButton** del componente RadioGroup (solo especifica el identificador de la opción).

  En tiempo de ejecución, usa el método **isCheck()** de un radio button para saber si está seleccionado. También, desde el grupo puedes obtener el identificador del RadioButton que tiene el estado On en tiempo de ejecución a través del método **getCheckedRadioButtonId()**.

Ejercicio 4: Crear una aplicación con las opciones «Depósito directo» y «Paypal». Al pulsar el botón "Guardar" si está seleccionada la opción "Depósito directo" imprime el mensaje *«Comprobar ubicación del usuario»* con un Toast.

- **Chips**: Las Chips son elementos de selección que permiten introducir información, realizar selecciones, filtrar contenido o ejecutar acciones de forma compacta.

  A continuación, se muestran los atributos principales:
  - **android.text**: El texto que se muestra en el Chip.
  - **android.chipIcon**: Icono que se muestra junto al texto.
  - **android.chipIconVisible**: Controla la visibilidad del icono. Debe estar a true para que el icono que se muestra junto al texto sea visible.
  - **android.checkable**: Define si el Chip se puede marcar o desmarcar. **Debe tener valor true para que pueda marcarse**.
  - **android.checked**: Indica si el Chip está actualmente marcado.
  - **android.checkedIcon**: Icono que se muestra al hacer click
  - **android.checkedIconVisible**: Controla la visibilidad del checkedIcon

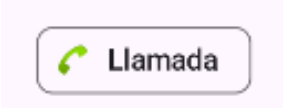
  Para usar Chips necesitas tener la dependencia de Material Components en tu build.gradle:

```
dependencies {
    implementation("com.google.android.material:material:1.12.0")
}
```

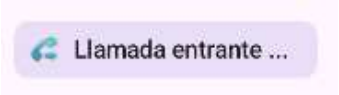
  Ejemplo: Al hacer click sobre el chip hacer que sea visible el icono configurado en checkedIcon. Para manejar el clic en el Chip, necesitas obtener una referencia al Chip en tu Activity o Fragment y configurar un OnClickListener.

```
val chip = binding.chip
chip.setOnClickListener {
    // Acción a realizar al hacer clic en el Chip
    if (chip.isChecked){
        chip.isCheckedIconVisible = true
        chip.text = "Llamada entrante ..."
    }
    else {
        chip.text = "Llamada"
    }
}
```

  Chip al cargar la pantalla:



  Chip al hacer click:

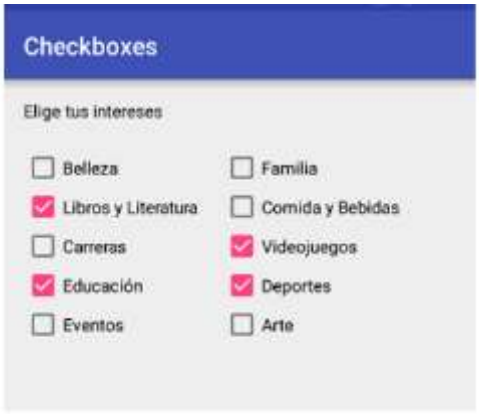


- **CheckBox**: Permite al usuario la selección de múltiples opciones. Un Checkbox es un botón de dos estados (marcado, no marcado) ante los usuarios. Lo que permite elegir una o varias opciones de un conjunto.

  Es similiar al RadioButton, salvo que permite seleccionar múltiples opciones, en lugar de una sola.

Ejercicio 5: En una Activity, añade un campo password y un checkbox. Revela el password que el usuario ha escrito al marcar el CheckBox.

Ejercicio 6: Crea un layout para presentar al usuario un conjunto de gustos que puede seleccionar para recomendar información de acuerdo a sus preferencias. Añade un botón y al pulsarlo muestra las opciones seleccionadas en un mensaje temporal (Toast).



Ejercicio 7: Crea un **formulario** para recoger los siguientes datos de una persona, eligiendo las TextInputLayout apropiadas y dando un buen aspecto.

- Nombre, Apellido1, Apellido 2, Teléfono (sólo números 9 dígitos), Fecha de nacimiento, email (formato correcto), password (2 campos y tienen que ser iguales) y un campo observaciones.
- Además, habrá 2 botones, uno Validar y otro Limpiar.
- Pon alguna imagen en la ventana.
- Cambia el fondo e icono de la aplicación.
- La aplicación debe ser válida para 2 idiomas y todos los colores creados mediante recursos.
- Importante dar un id significativo a los controles.
- Si se pulsa Validar, se comprobarán que se cumplen los requisitos de datos del formulario, y si es así, se llamará a una segunda ventana donde se mostrarán cada uno de los datos enviados (usa textView para mostrar los datos). Si hay algún error de validación, informa en la TextInputLayout indicada del error de validación ocurrido.
- Si se pulsa limpiar, se dejarán todos los campos vacíos.