Kotlin

MINI APUNTES ANDROID 2

Programación Multimedia y Dispositivos Móviles

3 DE NOVIEMBRE DE 2024

ERIK AMO TOQUERO

Contenido

Vistas

Las vistas son los distintos componentes que forman un layout. Todos ellos descienden de View, tienen un onTapListener, un ID y se puede acceder a ellos con el uso de findViewById<TipoView>

Todos ellos tienen una serie de parámetros en común:

- **Width y Height:** Tamaño de las vistas (Anchura y Altura)
- **End, Top, Bottom y Start:** Son los distintos puntos de referencia que tiene cada vista (Derecha, Arriba, Abajo e izquierda respectivamente). Permiten establecer su conexión y posición respecto a otras views de la tabla
- **Id:** El identificador de la vista, la referencia que se va a usar para llamarla (Puede estar vacío si no se usa, pero no es recomendable)
- **Visibility:** Si se puede o no ver la vista.

Ya de ahí puede haber muchas más variables que toman las vistas, teniendo más o menos importancia dependiendo de qué vista sea

Tipos de Vista

Como hay muchos voy a pararme en los que hemos dado en clase:

Button – Botón

Los botones son los principales views que realizan acciones. En el apartado visual, pueden tener los distintos recursos:

- **Text:** Texto del botón
- **Clickable:** Si se puede accionar o no
- **onAction:** Función que recorre cuando se activa

Esta última opción es la que normalmente se va a realizar de forma programática en la actividad, ya que causa menos errores en general

```
button.setOnClickListener(){
    [...]
}
```

TextView – Label de FX

Es el puro y duro equivalente al Label de JavaFX, es un contenedor de texto plano que no permite edición ni nada por el estilo, solo texto.

Permite editar su texto, eso sí, con setText(), o con text en Kotlin.

```
textView.text = "Hola Mundo"
```

EditText – Texto Editable

Este engloba todo tipo de contenedores de texto, ya que, con su variable **inputType**, se le puede cambiar el tipo de dato que está destinado a mandar (phone, email, password, multilineInput...). Contiene también la variable **hint**: Un texto guía que se puede poner y mostrar cuando el EditText está vacío.

Dentro de la parte de programación, resaltan los métodos de getText() y setText(String) (resumidos en text) y el método **addTextChangedListener(TextWatcher).** Este sirve para crear un método encargado de ver el estado del texto antes de, durante y después de editarlo.

```
editText.addTextChangedListener(object:TextWatcher{
    override fun beforeTextChanged(s: CharSequence?, start: Int,
count: Int, after: Int) {
        //Método que mira antes de ser cambiado
    }

    override fun onTextChanged(s: CharSequence?, start: Int, before:
Int, count: Int) {
        //Método que mira mientras se cambia
    }

    override fun afterTextChanged(s: Editable?) {
        //Método que mira tras cambiarse
    }
})
```

ImageView

Contiene un objeto de tipo Image que muestra al usuario. Dicha imagen puede salir de un recurso local (Drawable) o de la red. Normalmente se utiliza su método setVisible y setImageX, siendo X los distintos tipos de fuentes de los que puede sacar imágenes:

```
setImageResource(resId: Int)
setImageURI(uri: Uri?)
setImageIcon(icon: Icon?)
setImageLevel(level: Int)
setImageBitmap(bm: Bitmap!)
setImageDrawable(drawable: Drawable?)
setImageState(state: IntArray!, merge: Boolean)
```

Switch

Es un botón con cambios de estado. Interesa su atributo booleano isChecked, que permite ver si se ha pulsado o no

```
if (switch.isChecked){
    [...]
}
```

CheckBox

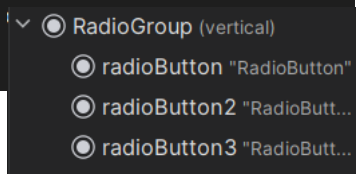
Funciona igual que el switch, su atributo más importante es isChecked

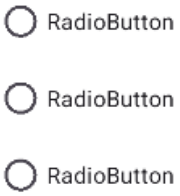
```
if (checkBox.isChecked){
    [...]
}
```

RadioButton y RadioGroup

Los RadioButton por separado tienen un funcionamiento similar a los CheckBox y a los Switch, pero cuando se juntan en un RadioGroup permiten hacer una selección de uno entre N elementos, siendo N el número de botones en el grupo.

```
var radioGroup:RadioGroup;
var radioButton =
findViewById<RadioButton>(radioGroup.checkedRadioButtonId)
when(radioButton.text){
    [...]
}
```





ToggleButton

Otra forma de checkBox. Sirve también para verificar si ha sido o no pulsado. Cuenta con

```
if (toggleButton.isChecked){
    [...]
}
```

Layouts

Los layouts contienen las vistas y grupos de vistas de una actividad. Se pueden crear de dos formas: En un xml con un editor gráfico o de texto, y de forma programática en la propia clase de la actividad (Está en desuso)

Pueden ser de varios tipos:

- **ConstraintLayout:** Da mucha libertad de dónde poner las vistas, teniendo que ponerlas restricciones (constraints) para dar su posición en la pantalla.
- **LinearLayout:** Distribuye los elementos uno tras otro. Cuenta con layout_gravity (separación entre elementos) y layout_weight (“peso” que tiene los componentes entre ellos, define su tamaño en relación a los demás). Tiene distintas orientaciones
    - **Vertical:** Vista de columna
    - **Horizontal:** Vista en fila
- **TableLayout:** Muestra los elementos en una tabla
- **FrameLayout:** Posiciona las vistas usando todo el contenedor, dividiéndolo en x número de secciones (normalmente 9)

Cambio entre Activities

Para cambiar entre actividades, se utilizan los Intents.
Un Intent es un elemento de Android Studio que se encarga de usar elementos externos a la actividad principal. Uno de sus usos es el de cargar actividades y pasar de una a otra.

Sintaxis de cambio de Activity: intent = Intent(context, Actividad2::class.java); startActivity(intent);

Tenemos una aplicación con dos pantallas: Una de ellas con un botón y un TextField y la otra con un TextView. Lo primero que se va a hacer es simplemente que, cuando se de al botón, salte de una pantalla a la otra

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_main)
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets -
>
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right,
systemBars.bottom)
            insets
        }
        var button:Button = findViewById(R.id.button)
        var eText:EditText = findViewById(R.id.texto);
        button.setOnClickListener{
            val intent:Intent = Intent(this, SecondActivity::class.java)
            startActivity(intent)
        }
    }
}
class SecondActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_second)
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets -
>
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right,
systemBars.bottom)
            insets
        }
        var saludoTView:TextView = findViewById(R.id.saludoTView)
    }
}
```

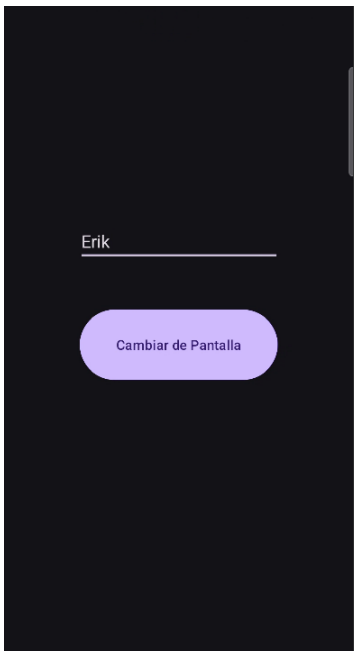
Resultado:

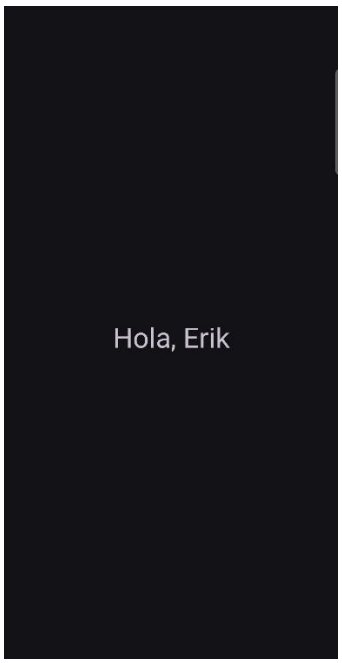
Cambio con paso de datos:

Se pueden pasar datos de la pantalla 1 a la 2 con intent.putExtra e intent.getExtra:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_main)
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) {
v, insets ->
            val systemBars =
insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right,
systemBars.bottom)
            insets
        }
        var button:Button = findViewById(R.id.button)
        var eText:EditText = findViewById(R.id.texto);
        button.setOnClickListener{
            val intent:Intent = Intent(this, SecondActivity::class.java)
            intent.putExtra("Nombre", eText.text.toString()) //Guarda el valor
            startActivity(intent)
        }
    }
}
class SecondActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_second)
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) {
v, insets ->
            val systemBars =
insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right,
systemBars.bottom)
            insets
        }
        var saludoTView:TextView = findViewById(R.id.saludoTView)
        var nombre:String = intent.getStringExtra("Nombre").toString();
//Devolvería String?
        if (nombre!=null){
            saludoTView.text = "Hola, $nombre"
        }
    }
}
```

Resultado:





Listeners

Los listeners son objetos creados para esperar eventos concretos y ejecutar una función cuando salten. En nuestro caso, solo hemos dado 2: OnClickListener y OnTextChangedListener:

- **OnClickListener:** Ejecuta una función cuando la vista es pulsada
- **OnTextChangedListener:** Cuando el texto de una vista cambia, un objeto de TextWatcher ejecuta 3 acciones: una antes de la edición, otra durante la edición y otra después

Uso de recursos

Los recursos guardados en la carpeta res se pueden usar de 2 formas:

- En formato xml en otros recursos
- Usando la clase R en código

Tipos de Recursos

- **Values:** Valores simples dentro del directorio values. Se accede a ellos con @tipo/nombrevalor en XML y getValue(R.tipo.nombre) en código
- **Colors:** Identifica colores. Se accede a ellos en XML como @colors/nombre y en Kotlin con resources.getColor(R.color.nombre) (Devuelve un entero que corresponde al código ARGB del color)
- **Estilos:** Valores de estilos para las vistas, solo se pueden leer en XML con @style/nombre
- **ID:** Identificadores de vistas, se leen con @id/nombre o con R.id.nombre
- **Arrays:** Se leen en código con resources.obtainTypedArray(R.array.nombre) o resources.getInt/StringArray(R.array.nombre) para arrays de enteros o de strings respectivamente