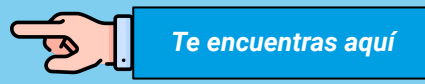
Kotlin para el desarrollo de aplicaciones

Kotlin y Android (Parte II)

{desafío} latam_

Reconocer las principales características del lenguaje Kotlin para el desarrollo de aplicaciones móviles Android Nativo.

- Unidad 1: Kotlin para el desarrollo de aplicaciones.



- Unidad 2: Ciclo de vida de componentes.
- Unidad 3: Arquitectura en Android.
- Unidad 4: Programación asíncrona en Android.

{desafío} latam_

¿Qué aprenderás en esta sesión?

- *Codificar una aplicación básica utilizando Event Listeners y View Bindings acorde al lenguaje Kotlin*

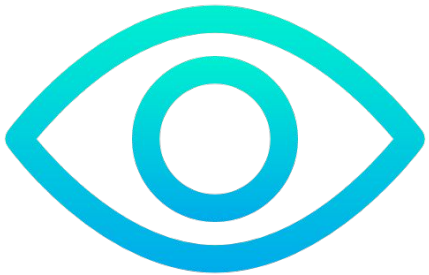
{desafío} latam_

Recordemos...

¿Qué sabemos sobre nulidad y problemas NPE?

/* View Binding */

View Binding o Vinculación de vista



La vinculación de vista es una función que te permite escribir más fácilmente código que interactúa con las vistas. Una vez que la vinculación de vista está habilitada en un módulo, genera una clase de vinculación para cada archivo de diseño XML presente en ese módulo. Una instancia de una clase de vinculación contiene referencias directas a todas las vistas que tienen un ID en el diseño correspondiente.

En la mayoría de los casos, la vinculación de vistas reemplaza a `findViewById`.

Fuente: https://developer.android.com/topic/libraries/view-binding

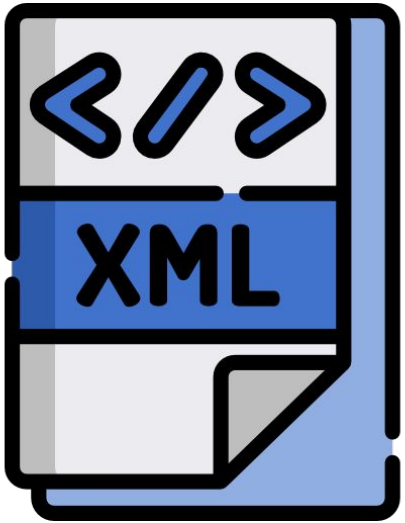
Habilitando View Binding

Para habilitar view binding en un proyecto Android se debe configurar el archivo **`gradle`** correspondiente a cada módulo en el que se desee ocupar.

```
android {
    ...
    buildFeatures {
        viewBinding = true
    }
}
```

Tip: En las últimas versiones de Android Studio, view binding viene habilitado por defecto.

¿Cómo usar View Binding?



Cuando se habilita view binding en un módulo, este generará una clase de vinculación para cada vista de diseño (XML).

Cada clase de vinculación contiene referencias a la vista raíz y a todas las vistas que tienen un ID. El nombre de la clase de vinculación se genera convirtiendo el nombre del archivo XML según la convención de mayúsculas y minúsculas, y agregando la palabra "**Binding**" al final.

¿Cómo usar View Binding?

Por ejemplo, si creamos un activity llamado **`UserDetailActivity`**, la vista de diseño de ese activity debería llamarse **`activity_user_detail.xml`** y el archivo de vinculación se llamará:

`ActivityUserDetailBinding`.

Para el caso de Fragments, solo se reemplazará la palabra activity por fragment, por ejemplo:

`FragmentUserDetailBinding.`

¿Cómo usar View Binding en Activities ?

Supongamos que creamos un Activity y lo nombremos `MainActivity` y a la vista de diseño XML, la nombramos `activity_main`:

Definimos una variable lateinit tipo **`ActivityMainBinding`**:

```
private lateinit var binding:
ActivityMainBinding
```

Dentro de onCreate, inicializamos binding de la siguiente forma:

```
binding =ActivityMainBinding
.inflate(layoutInflater)
```

Finalmente, para usar binding lo podemos hacer de la siguiente forma:

```
binding.fab.setOnClickListener { view ->
Snackbar.make(view, "pressed!",
Snackbar.LENGTH_LONG).show()
    }
```

Donde fab es un `FloatingActionButton` definido en la vista de diseño

```
<com.google.android.material.floatingactionbutton.Fl
oatingActionButton
android:id="@+id/fab"
... />
```

¿Cómo usar View Binding en Fragments ?

Para usar view binding en un Fragment:

Creamos una variable tipo `FragmentFirstBinding?` con valor inicial nulo

```
private var _binding: FragmentFirstBinding? = null
private val binding get() = _binding!!
```

Luego, en `onCreateView` inicializamos la variable `_binding` y retornamos `binding.root`

```
_binding = FragmentFirstBinding.inflate(inflater,
container, false)
return binding.root
```

A diferencia del Activity, en Fragment es necesario sobreescribir el método `onDestroyView` y hacer `_binding` nulo:

```
override fun onDestroyView() {
    super.onDestroyView()
_binding = null
}
```

{desafío} latam_

Ventajas de View Binding

La vinculación de vistas tiene ventajas importantes frente al uso de `findViewById`:

Null Safety

Debido a que la vinculación de vista crea referencias directas a las vistas, no hay riesgo de una excepción de puntero nulo debido a un ID de vista no válido. Además, cuando una vista solo está presente en algunas configuraciones de un diseño, el campo que contiene su referencia en la clase de vinculación se marca con `@Nullable`.

Type safety

Los campos de cada clase de vinculación tienen tipos que coinciden con las vistas a las que hacen referencia en el archivo XML. Esto significa que no hay riesgo de una excepción de transmisión de clase.

Estas diferencias significan que las incompatibilidades entre tu diseño y tu código harán que falle la compilación durante el momento de compilación en lugar de hacerlo en el tiempo de ejecución.

findViewById

En un par de ocasiones hemos mencionado `findViewById`, antes de view binding se utilizaba `findViewById` para conectar la vista de diseño con la lógica.

Una de las principales desventajas era que permitía conectar con algún objeto dentro de la vista de diseño, pero que no estaba relacionado al activity o fragment, ya que sólo validaba el tipo.

Por ejemplo, podía llamar desde MainActivity

```
val fab: FloatingActionButton = findViewById(R.id.fab),
```

Sin necesidad de estar ahí, y solo validando que **`fab`** es de tipo **`FloatingActionButton`**

/* Arrow functions para event listeners */

Arrow operator

El operador flecha o Arrow cumple tres funciones:

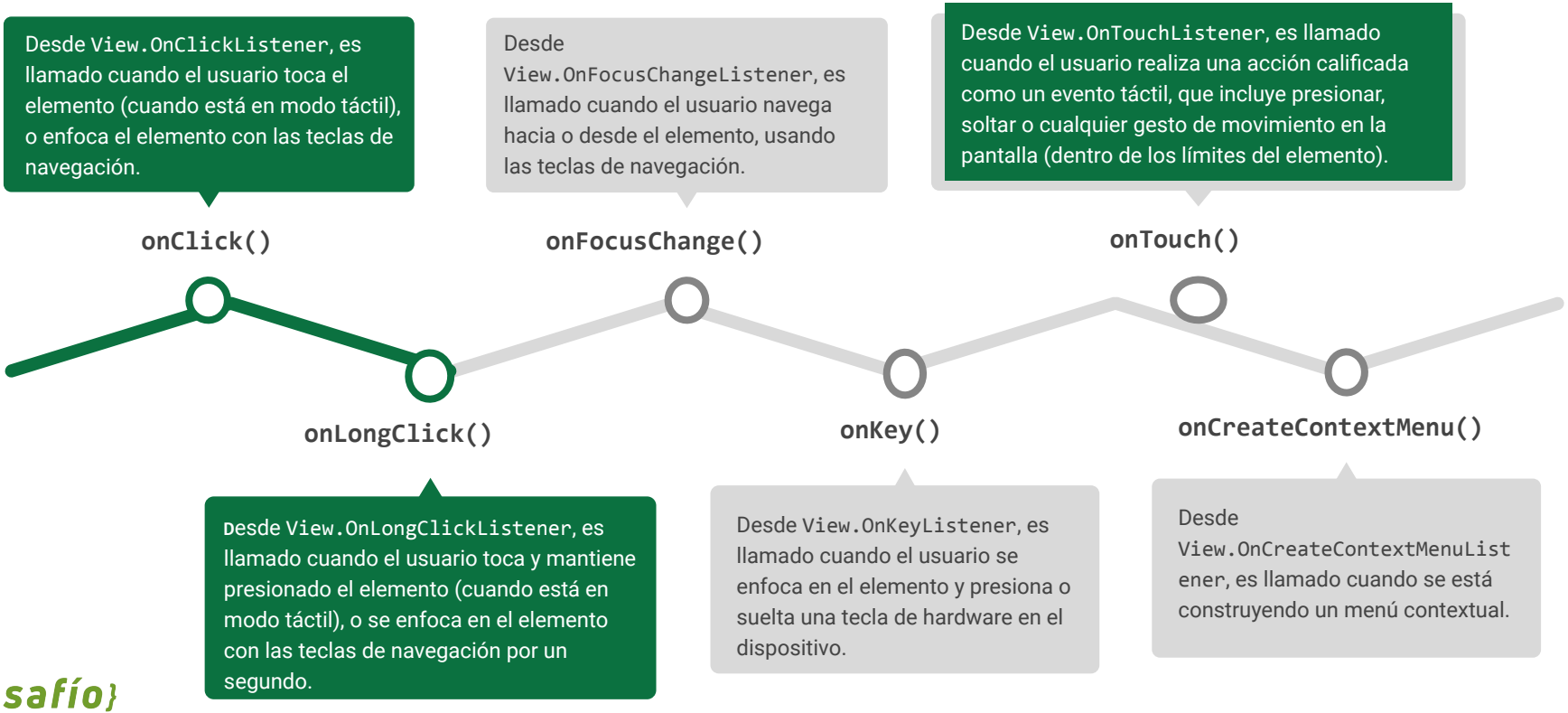
- Separa los parámetros y el cuerpo de una expresión lambda.
- Separa los parámetros y la declaración del tipo de retorno en un tipo de función.
- Separa la condición y el cuerpo de una rama de expresión **`when`**.

Event listener

En Android, existe más de una forma de interceptar los eventos desde una interacción del usuario con tu aplicación. Al considerar los eventos dentro de tu interfaz de usuario, el enfoque consiste en capturar los eventos desde el objeto de vista específico con el que interactúa el usuario.

Un Event Listener es una interfaz en la clase **`View`** que contiene un único método de devolución de llamada. El marco de trabajo de Android llamará a estos métodos cuando la vista en la que se ha registrado el oyente se activa por la interacción del usuario con el elemento en la interfaz de usuario.

Event listener - Callbacks



{desafío} latam_

Ejemplo Event listener - Callbacks

A continuación se muestran algunos ejemplos de ¿cómo usar estos callbacks?

En este ejemplo se muestra cómo registrar el evento on-click para un FAB (`FloatingActionButton`):

```
binding.fab.setOnClickListener { view ->
// tu código aquí
}
```

Otra forma es implementando `View.OnClickListener`, ejemplo:

```
class MainActivity : AppCompatActivity(),
View.OnClickListener
override fun onCreate(savedInstanceState:
Bundle?) {
    super.onCreate(savedInstanceState)

    binding.fab.setOnClickListener(this)

fun onClick(v: View) {
    // tu código aquí
}
```

Ejemplo Event listener - Callbacks

Se puede usar más de un callback, si es necesario, por ejemplo usamos **`onClick`** y **`onLongClick`**:

```
class MainActivity : AppCompatActivity(), View.OnClickListener,
View.OnLongClickListener
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
  binding.fab.setOnClickListener(this)
  binding.fab.setOnLongClickListener(this)

override fun onClick(view: View?) { /* tu código aquí */ }
override fun onLongClick(view: View?): Boolean { /* tu código aquí */ return
true}
```

Reflexionemos…

¿Qué es el View Binding?
¿Para qué nos sirve?

Reflexionemos…

¿Recuerdas las ventajas de View Binding? Nombra alguna

Reflexionemos…

Nombra 3 Event Listeners y para qué se utilizan

{desafío} latam_

Próxima sesión...

- *Codificar una aplicación básica utilizando Event Listeners y View Bindings acorde al lenguaje Kotlin*

{desafío}
latam_

Academia de
talentos digitales