Guía de ejercicios - Kotlin para el desarrollo de aplicaciones (II)

¡Hola! Te damos la bienvenida a esta nueva guía de estudio.

¿En qué consiste esta guía?

La siguiente guía de estudio tiene como objetivo practicar y ejercitar los contenidos que hemos visto en clase.

Tabla de contenidos

¡Comencemos!

Actividad guiada: Aplicando clases abstractas en Android

En la siguiente actividad guiada crearemos una clase base para nuestros activities. Una de las principales razones para hacer esto es que cuando se genera una aplicación que contiene varios activities, se repite código innecesariamente.

1. Crear un nuevo proyecto en Android Studio, seleccionar **Empty Activity** y continuar.

2. Dentro del package que contiene el **`MainActivty`**, crear un nuevo paquete y llamarlo "**`base`**"

3. Dentro del paquete "**`base`**" crear una nueva clase de Kotlin y llamarla **`BaseActivity`**

4. Transformar en clase abstracta agregando "**abstract**" antes de class

5. En Android los activities extienden de **`AppCompatActivity()`**, así que hacemos que **`BaseActivity`** también lo haga.

6. Ya con esto podemos extender de BaseActivity en vez de **`AppCompatActivity`**

7. Ahora le daremos un poco más de sentido a nuestra clase base, aplicando lo aprendido en la clase *Kotlin y Android (Parte II)*, View Binding. Haremos que cada vez que creamos un nuevo activity, no tengamos que definir paso a paso nuestro view binding.

8. Primero debemos habilitar view binding, en nuestro proyecto. Una vez habilitado view binding, modificamos nuestra clase base de forma que puede recibir un parametro tipo ViewBinding:

```
abstract class BaseActivity<T : ViewBinding> : AppCompatActivity()
```

   Esto generará un error en `MainActivity` y lo corregiremos en un momento.

9. Luego definimos una variable `lateinit` tipo genérico `T`

```
lateinit var binding: T
```

10. A continuación creamos una función abstracta que retorne un tipo genérico T, cuando esta función se use en el activity retornara el view binding específico de ese activity

```
abstract fun getViewBinding(): T
```

11. La función `onCreate` se llama en cada Activity y es donde se define la vista correspondiente a cada Activity. Ya que nuestra clase base es capaz de manejar view binding, moveremos la función **`onCreate`** del **`MainActivity`** a la clase **`BaseActivity`**

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

12. dentro de **`onCreate`**, tenemos que hacer que la variable **`binding`** que definimos anteriormente tome el valor de la función **`getViewBinding()`** y pasarle **`binding.root`** a `setContentView()`, quedando como en el siguiente snippet:

```
binding = getViewBinding()
setContentView(binding.root)
```

13. Finalmente, la clase BaseActivity debería verse de la siguiente forma:

```
abstract class BaseActivity<T : ViewBinding> : AppCompatActivity(){
    lateinit var binding: T

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = getViewBinding()
        setContentView(binding.root)
    }

    abstract fun getViewBinding(): T
}
```

14. Ahora en nuestro MainActivity debería haber varios errores, los que vamos a corregir.

```
class MainActivity : BaseActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

15. Primero, cambiamos la forma en que extiende de **BaseActivity** y le pasamos el view binding correspondiente a **MainActivity**

```
class MainActivity : BaseActivity<ActivityMainBinding>() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

16. Al colocar el cursor sobre el error, Android Studio indica que se deben implementar algunos miembros, aceptamos los cambios y ahora MainActivity debería verse de la siguiente forma:

```
class MainActivity : BaseActivity<ActivityMainBinding>() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }

    override fun getViewBinding(): ActivityMainBinding {
        TODO("Not yet implemented")
    }
}
```

17. Hacemos que **`getViewBinding()`** retorne el view binding y removemos la función onCreate, ya que fue definida en la clase base. Ahora nuestro MainActivity está terminado.

```
class MainActivity : BaseActivity<ActivityMainBinding>() {

    override fun getViewBinding(): ActivityMainBinding =
```

```
        ActivityMainBinding.inflate(layoutInflater)
}
```

¡Manos a la obra! - Usando View Binding

1. Crea un proyecto en Android Studio y selecciona “No Activity”, luego habilita el proyecto para trabajar con View Binding.

2. Crea un Activity incluida su vista de diseño (XML), en el package “**noactivity**”, nómbralo HomeActivity, el archivo xml debe ir en `res/layout`

3. En `activity_home.xml` agrega un botón, nómbralo **`btn_home`** y céntralo en medio de activity, puedes agregarle el texto que desees.

4. Configura tu view binding en `HomeActivity` para enlazarlo al botón

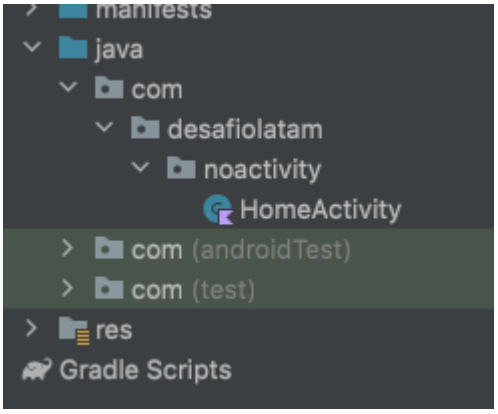
5. Finalmente, cuando presiones el botón, muestra un Toast con el mensaje “¡Hola Android!”

Respuestas Actividad 1 - Setting up View Binding

1. Configuración de View Binding en gradle

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

2. Screenshot de ejemplo:



3. Botón centrado

```
<androidx.appcompat.widget.AppCompatButton
        android:id="@+id/btn_home"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/app_name"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
```

4. HomeActivity terminado

```
class HomeActivity : AppCompatActivity() {

    lateinit var binder: ActivityHomeBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binder = ActivityHomeBinding.inflate(layoutInflater)
        setContentView(binder.root)
    }
}
```

5. Código con mensaje Toast

```
binder.btnHome.setOnClickListener {
    Toast.makeText(this, "¡Hola Android!", Toast.LENGTH_LONG).show()
}
```

Preguntas de proceso

Reflexiona:

- A partir de lo aprendido hasta aquí, ¿Qué contenido se me ha hecho más sencillo? ¿Y cuál más difícil?
- ¿Existe algún ejercicio de los resueltos previamente que deba repetir o desarrollar otra vez para poder dominarlo mejor?
- Nombra al menos un uso que pueda darle a lo aprendido hasta ahora.
- ¿En qué situación se podría ocupar lo aprendido hasta el momento?

Preguntas de cierre

- ¿Cuál puede ser una ventaja de las clases abstractas?
- ¿Cuál es la diferencia entre una función lambda y una interfaz anónima?
- ¿Para qué se utiliza el Companion Object?
- Nombre una ventaja de View Binding sobre `findViewById`
- ¿Cómo puedo hacer para trabajar con dos clases abstractas?

Referencias bibliográficas

- **Documentación**: https://kotlinlang.org