INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

INGENIERÍA EN SISTEMAS COMPUTACIONALES

MATERIA: APPLICATION DEVELOPMENT FOR MOBILE DEVICES

PROFESOR: CIFUENTES ALVAREZ ALEJANDRO SIGFRIDO

PRESENTA:

RAMIREZ BENITEZ BRAYAN

GRUPO: 3CM17

“TAREA – Kotlin Parte 2”

CIUDAD DE MEXICO A 3 DE MAYO DE 2022

Null

En Kotlin, todas las variables se consideran no anulables de manera predeterminada, por lo que al intentar asignar un valor null a una variable dará como resultado un error de compilación. Por ejemplo, lo siguiente no se compilará:

var ejemplo: String = null

Si desea declarar explícitamente que una variable puede aceptar un valor null, entonces deberá agregar un ? al tipo de variable. Por ejemplo, se compilará lo siguiente:

var ejemplo: String? = null

```
//Variable Null
var a : String? = null
```

Operador ?

El operador ? de llamada segura ofrece una forma de tratar con las referencias que potencialmente podrían contener un valor null, al tiempo que garantiza que cualquier llamada a esta referencia no dará como resultado una NullPointerException.

Por ejemplo, lo siguiente devolverá a.size solo si a no es nulo; de lo contrario, devolverá null: a?.size

También puede encadenar operadores de llamada segura juntos:

val number = person?.address?.street?.number

```
//Operador ?
Log.d( tag: "tag: ",  msg: ""+a?.length)
```

El operador Elvis ?:

Algunas veces se podría tener una expresión que potencialmente podría contener un valor nulo, pero sin lanzar una NullPointerException, incluso si este valor resulta ser null.

En estas situaciones, se puede usar el operador Elvis ?: de Kotlin para proporcionar un valor alternativo que se usará cada vez que el valor resulte nulo, lo cual es una buena forma de evitar la propagación de valores nulos en el código. Por ejemplo: fun setName (name: String?) { username = name?: "N/A" }

Aquí, si el valor a la izquierda del operador de Elvis no es nulo, entonces se devuelve el valor de name del lado izquierdo.

Pero si el valor a la izquierda del operador de Elvis es null, se devolverá la expresión de la derecha, que en este caso es N/A.

```
//Operador ?:
fun setColor(color:String?){
    var c = color?:"N/A"
    var a : String? = "El color es: " + c
}
```

El operador !!

Si alguna vez se quiere obligar al código de Kotlin a lanzar una NullPointerException de estilo Java, se puede usar el operador !!. Por ejemplo: val number = firstName!!.length

Aquí, se utiliza el operador !! para afirmar que la variable firstName no es nula.

Siempre que firstName contenga una referencia válida, el valor de la variable se asigna con la longitud de la cadena. Si firstName no contiene una referencia válida, entonces Kotlin lanzará una NullPointerException.

```
//Operador !!
if(a!!.toString().equals("")){
    setColor(null)
}else{
    setColor("verde")
}
```

Expresión Lambda y Funciones de extensión

Una expresión lambda representa una función anónima. Las lambdas son una buena forma de reducir la cantidad de código necesario para realizar tareas en el desarrollo de Android; por ejemplo, escribir escuchas y devoluciones de llamada. Java 8 ha introducido expresiones lambda nativas, y ahora son compatibles con Android Nougat, aunque esta característica no es exclusiva de Kotlin pero se pueden usar en todo el proyecto.

Una expresión lambda consta de un conjunto de parámetros, un operador lambda (->) y un cuerpo de función, dispuestos en el siguiente formato: {x: Int, y: Int -> x + y}

Al igual que en C#, Kotlin permite agregar nuevas funcionalidades a las clases existentes que de otro modo no se podrían modificar. Entonces, si a una clase le falta un método útil, se le puede agregar a través de una función de extensión.

Para crear una función de extensión se prefija el nombre de la clase que se desea extender con el nombre de la función que se está creando. Por ejemplo:

fun AppCompatActivity.toast(msg: String){

Toast.makeText(this, msg, Toast.LENGTH_LONG).show()

}

```
//Expresion Lambda y funcion de extension
boton.setOnClickListener {  it: View!

    //Operador !!
    if(a!!.toString().equals("")){
        setColor(null)
    }else{
        setColor("verde")
    }
}
```

Objeto Singleton

En Java, la creación de singletons ha sido típicamente muy detallada, lo que requiere que se cree una clase con un constructor privado y luego crear esa instancia como un atributo privado.

En lugar de declarar una clase, Kotlin permite definir un sólo objeto, que es semánticamente el mismo que un singleton, en una línea de código:

object KotlinSingleton {}

A continuación, se puede utilizar este singleton de inmediato, por ejemplo:

object KotlinSingleton {

fun myFunction (){ [...] }

}

KotlinSingleton.myFunction ()

```
//Objeto Singleton
kotlinSingleton.myFunction( context: this, msg: "Objeto s", Toast.LENGTH_LONG)
```

```
//Objetos singleton
object kotlinSingleton{
    fun myFunction(context: Context, msg:String, int: Int){
        Toast.makeText(context,msg,int).show()
    }
}
```

CONCLUSIONES

Los conceptos avanzados de Kotlin nos permiten hacer más cosas con menos líneas de código, así como también simplificar varios conceptos que en Java existen pero que se implementan de otra manera.