Tema 3 Kotlin intermedio

Índice

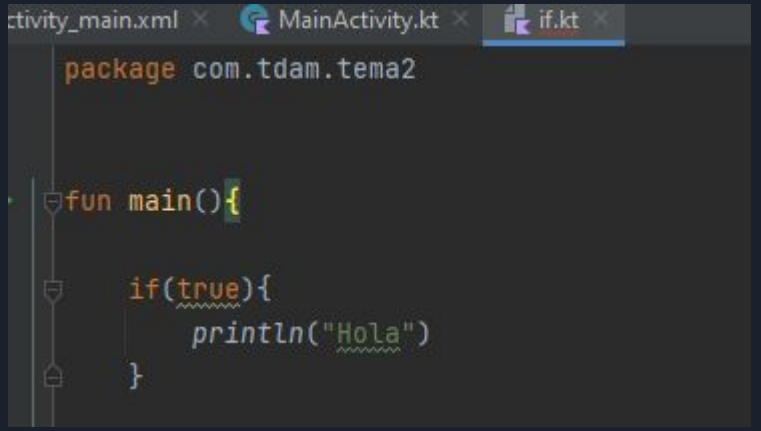
- Condicionales
  - If else
  - Equals() y ‘==’
  - Operadores lógicos
  - When
- Arrays
  - Vararg
  - Array
  - List
  - MutableList
  - Maps
  - Array de nulos
  - Métodos en colecciones
- Bucles
  - For
  - Foreach
  - While
  - Do while
- Return y break
- Ejercicios

Condicionales

If

A tener en cuenta:

El if de kotlin no permite comparar números




```
if(1) { //fallo
    println("Hola")
}
```

Equality y Equals

Equality

```
if (1 == 1) {
    println("Hola")
}
```

Equals

```
//Equals
if (1.equals(1)) {
    println("Hola")
}

val palabra = "Pedrito"

if(palabra.equals("Pedrito")){
    println("Hola")
}
```

Operadores lógicos

!= NO LÓGICO

```
//operadores logicos

if( 1 != 2){
    println("Hola")
}
```

> MAYOR QUE

```
if( 1 > 2){
    println("Hola")
}
```

Operadores lógicos

< MENOR QUE

```
if( 1 < 2){
    println("Hola")
}
```

>= MAYOR O IGUAL QUE

<= MENOR O IGUAL QUE

```
// >= <=
if( 1 <= 2){
    println("Hola")
}
if (3 >= 2) {
    println("Hola")
}
```

Operadores lógicos

|| OR

```
if( 1 > 0 || 1 < 2){
    println("Hola")
}
```

&& AND

```
if( 1 > 0 && 1 < 2){
    println("Hola")
}
```

If else

```
var a = 5
var b = 10

if(a > b){
    println("a es mayor que b")
}else{
    if(a == 5){
        println("a es igual a 5")
    }
}
```

When

```
//when

var c = 1

when(c){
    1 -> println("a es 1")
    2 -> println("a es 2")
    3 -> println("a es 3")
    else -> println("a es otro valor")
}
```

When también admite intervalos o condiciones en vez de valores

```
return when (calificacion) {
    in 90..100 -> "A"
    in 80 until 90 -> "B"
    in 70 until 80 -> "C"
    in 60 until 70 -> "D"
    in 0 until 60 -> "F"
    else -> "Calificación no válida"
```

Arrays

Varargs

Con esto podemos pasarle a una función múltiples parámetros de forma dinámica

Bien podemos pasarle 3 o 1

A diferencia de otros parámetros que no necesitan declarar que tipo de variable son, si es un 'vararg' debe ponerse delante del nombre de la variable

```
args en las posicion 0 es hola
hola
chau
hola2
```

```
    variosArgs( ...args: "hola", "chau","hola2")


}

fun variosArgs(vararg args: String){

    println("args en las posicion 0 es ${args[0]}")

    for (arg in args){
        println(arg)
    }
}
```

Declarada fuera del main

Array

Array simple

```
//array simple
val array = arrayOf('h','o','l','a')
```

```
//dos formas de acceder al array
println(array[0])
println(array.get(0))
```

String array

```
//los strings son también arrays
val palabra2 = "hola"
println(palabra2[0])
```

Los elementos de los arrays suelen ser del mismo tipo aunque pueden ser de distintos

Admiten lectura y escritura

```
array[0] = 'a'
```

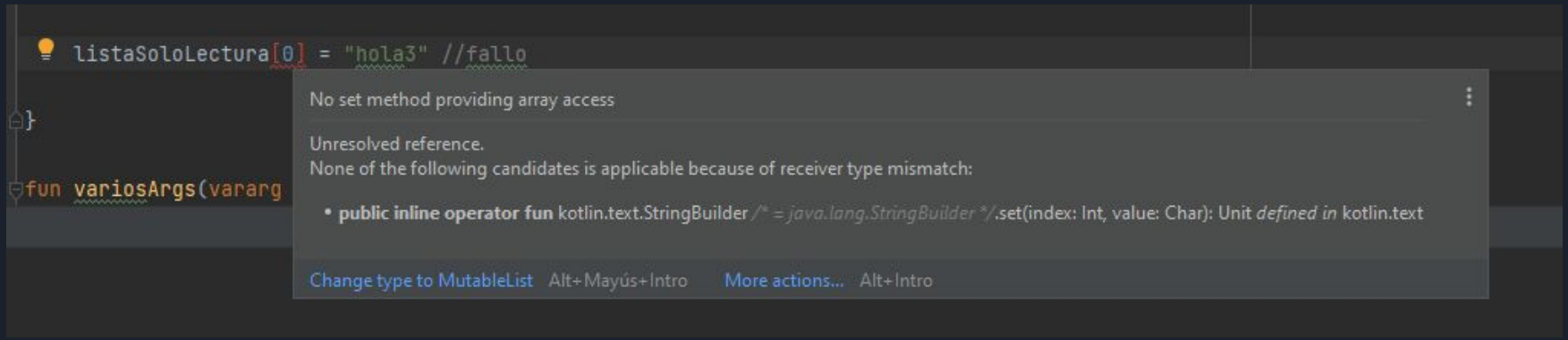
ListOf

Tenemos método get al ser una lista de elementos

Pero es una lista estática y solo vale para lectura

```
//ListOf

val listaSoloLectura = listOf("hola", "chau", "hola2")
println(listaSoloLectura[0])
println(listaSoloLectura.get(1))
```



mutableListOf

Permiten lectura y escritura

```
//podemos imprimir directamente el array
println(listaMutable)
```

```
[hola3, chau, hola2]
```

```
//mutableListOf

val listaMutable = mutableListOf("hola", "chau", "hola2")
println(listaMutable[0])
println(listaMutable.get(1))
listaMutable[0] = "hola3"
listaMutable.add("hola4") //agrega un elemento al final
listaMutable.add( index: 1, element: "hola5") //agrega un elemento en la posicion 1
listaMutable.removeAt( index: 1) //remueve el elemento en la posicion 1
listaMutable.remove( element: "hola4") //remueve el elemento "hola4"
listaMutable.set(0,"hola6") //setea el elemento en la posicion 0
```

Map

Funciona con Clave => valor

La ventaja es que el acceso a los elementos es más rápido que con otros arrays debido a que está optimizado para leer a través de la clave

La desventaja es que no podemos poner una posición directa tal cual poniendo map[1]

```
//mutableMap

val map = mutableMapOf<Int,String>() //creamos un map vacio de clave Int y valor String
map.put(1,"Uno") //clave, valor
map.put(2,"Dos")
map.put(3,"Tres")
map.remove( key: 2) //remueve el elemento con clave 2
map.set(1,"Uno") //setea el elemento con clave 1
println("Mapa: $map")
println("Mapa: ${map.get(1)}") //podemos acceder al valor de la clave 1

val map2 = mutableMapOf<String,String>()
map2.put("1","Uno") //clave, valor
map2.put("2","Dos")
println("Mapa 2: ${map2.get("1")}") //podemos acceder al valor de la clave "1"


println(map.keys)
println(map.values)
```

Acceder a claves y valores de mutableMap

```
println(map.keys)
println(map.values)
```

```
[1, 3]
[Uno, Tres]
```

Array de null

Aquí si necesitamos especificar la longitud porque únicamente estamos reservando ese espacio pero aún no sabemos cuáles serán los elementos

Nos vale si ya sabemos cuanto datos vamos a obtener por ejemplo en una división (resto y cociente)

```
//arrayOfNulls

val arrayNulls = arrayOfNulls<String>( size: 5) //creamos un array de 5 elementos nulos
arrayNulls[0] = "hola"
println(arrayNulls[0])
```

Métodos en colecciones

método sorted() y método reversed()

```
val lista = listOf("hola", "chao", "jeje")
//metodo sorted
println(lista.sorted()) //ordena la lista pero no la modifica
//metodo reversed
println(lista.reversed()) //invierte la lista
```

```
[chao, hola, jeje]
[jeje, chao, hola]
```

método indexOf() y método

```
//metodo indexOf
println(lista.indexOf("chao")) //devuelve el indice del elemento "chao"
```

```
1
```

Bucles

For

```
package com.tdam.tema2

fun main(){

    //bucle for

    bucle( ...args: "Pedro","Maria","Juan")

}
```

```
fun bucle(vararg args: String) {

    //forma con rango
    for (i in 1 ≤ .. ≤ 10) {
        println(i)
    }
    //forma de 2 en 2
    for (i in 1 ≤ .. ≤ 10 step 2) {
        println(i)
    }
    //forma con size
    for(i in 0 ≤ .. ≤ args.size-1){
        println(args[i])
    }

    //forma con <
    for(i in 0 ≤ until < args.size){
        println(args[i])
    }

    //forma normal
    for(name in args){
        println(name)
    }
```

forEach

```
//forEach
args.forEach { it: String
    println(it) //este nombre es el que se le da por defecto
}

//si queremos que sea otro nombre mas descriptivo
args.forEach { nombre ->
    println(nombre)
}

//rangos
(1 ≤ .. ≤ 10).forEach { it: Int
    println(it)
}
```

While

```
package com.tdam.tema2

fun main(){

    //bucle while
    val nombres = arrayListOf<String>("Pedro","Maria","Juan")
    var i = 0
    while(i < nombres.size){
        println(nombres[i])
        i++
    }

}
```

Do While

La diferencia entre el do while y el while es que uno ejecuta al menos una vez el cuerpo del bucle

```
// do while
var j = 0
do{
    println(nombres[j])
    j++
}while(j < nombres.size)
```

Return y Break

return a secas hace que se salga totalmente del método en el que esté, en este caso el main.

break permite que el código continue y solo se sale del ámbito en el que esté

return@"bucle" hace que acabe la sentencia pero pueda continuar por donde iba en el bucle

```
//return y break

(1 ≤ .. ≤ 5).forEach{ it: Int
    if(it == 3){
        return@forEach //esto es como un continue
    }
    println(it)
}
```

```
1
2
4
5
```

```
for (i in 1 ≤ .. ≤ 10) {
    if(i == 5){
        break
    }
    println(i)
}
```

```
1
2
3
4
```

Ejercicios

Condicionales:

1. Escribe una función llamada esPar que tome un número como argumento y devuelva true si es par y false si es impar.

2. Crea una función llamada compararTexto que tome dos cadenas de texto como argumentos y determine si son iguales utilizando tanto equals() como el operador ==.

3. Implementa una función llamada esMayorQue10YMenorQue20 que tome dos números como argumentos y devuelva true si ambos números son mayores que 10 y menores que 20, y false en caso contrario.

4. Crea una función llamada evaluarCalificacion que tome la calificación de un estudiante (un número entre 0 y 100) como argumento y devuelva una letra de calificación (A, B, C, D, F) utilizando una estructura when.

Arrays

5. Define una función llamada sumarNumeros que tome un número variable de argumentos utilizando vararg y devuelva la suma de todos los números.

6. Implementa una función llamada invertirArray que tome un array de números como argumento y devuelva un nuevo array con los elementos en orden inverso.

7. Crea una lista inmutable de nombres de frutas y agrega una fruta adicional a la lista.

8. Utiliza una lista mutable para almacenar nombres de colores y elimina un color de la lista.

Bucles

9. Escribe una función llamada imprimirNumeros que utilice un bucle for para imprimir los números del 1 al 10.

10. Define una función llamada imprimirLista que tome una lista de números como argumento y utilice un bucle forEach para imprimir cada número en la lista.

11. Implementa una función llamada imprimirNumerosHastaN que tome un número como argumento y utilice un bucle while para imprimir los números del 1 al número dado.

12. Crea una función llamada imprimirNumerosPositivos que utilice un bucle do-while para imprimir números positivos hasta que se alcance un número negativo.

Return y break

13. Define una función llamada esPrimo que tome un número como argumento y devuelva true si es primo y false si no lo es. Utiliza una declaración return para salir de la función una vez que se determine si el número es primo.

14. Implementa una función llamada encontrarPrimoMayorQue100 que utilice un bucle for para encontrar el primer número primo mayor que 100. Utiliza una declaración break para salir del bucle una vez que se encuentre el número primo.