Capturas de Kotlin en Clase

Tipos y Variables

Arriba la declaración de variables y abajo la salida

```
fun main() {
    var patata = "Antes, todo esto, era campo"//variable
    var texto:String = "Le Text"//Variable definiendo tipo cadena
    var numerete:Int = 33//variable definiendo tipo Entero
    var doblete: Double = 65.7//Variable Double
    var flotador: Float = 67.6f//Float debe llevar una f al final
    val cuack = "pato"//Constante
    println("Hello, world!!!")
    println("$patata $texto $numerete $doblete $flotador $cuack")
}
```

```
Hello, world!!!
Antes, todo esto, era campo Le Text 33 65.7 67.6 pato
```

Listas

```
fun main() {
    var listaCiclista: MutableList<String> = mutableListOf("Patata", "Zanahoria", "Claudio", "Pato") //Creamos lista y definimos contenidos de esta
    println(listaCiclista)//Imprimimos lista en pantalla
    println(listaCiclista[0])//Imprimimos en pantalla el primer elemento de la lista (El que sería el elemento 0)
    listaCiclista.add("Campo")//Añadimos elemento a la lista
    println(listaCiclista)
    listaCiclista.remove("Zanahoria")//eliminamos elementos de la lista
    println(listaCiclista)
    listaCiclista.count()//Cuenta el número de elementos de la lista
    println("Hay ${listaCiclista.count()}")//Mostramos en pantalla el número de elementos en la lista
}
```

```
[Patata, Zanahoria, Claudio, Pato]
Patata
[Patata, Zanahoria, Claudio, Pato, Campo]
[Patata, Claudio, Pato, Campo]
Hay 4
```

Conjuntos

```
fun main() {
    var conjuntoCiclistas: Set<String> = setOf("nokia","Ex","LeQuique")
    //funciona Igual que las Listas mas o menos

}
```

Mapas

Se pueden usar para almacenar datos de una forma similar a una base de datos

```
fun main() {
    //Map<Clave, ValorAlmacenado>
    val mapaCiclista: Map<String, Int> = mapOf("patata" to 1, "Zanahoria" to 2, "No se rick, parece falso" to 3)
    val mapaCiclista2: Map<Int, String> = mapOf(1 to "dani", 2 to "pepe", 3 to "cuacko")
    val mapaCiclista3: Map<Int, List<String>> = mapOf(1 to listOf("a","b","c","d"), 2 to listOf("d","e","f"))
    println(mapaCiclista)
    println(mapaCiclista2)
    println(mapaCiclista3)
    println(mapaCiclista2[2])//Ejemplo para acceder, por ejemplo, al valor pepe. Buscaríamos por clave
}
```

```
{patata=1, Zanahoria=2, No se rick, parece falso=3}
{1=dani, 2=pepe, 3=cuacko}
{1=[a, b, c, d], 2=[d, e, f]}
pepe
```

Flujos de Control

Bucle FOR

```
fun main() {
    val clasificacion: Map<Int, String> = mapOf(1 to "SeppKuss", 2 to "Kiam Auisp", 3 to "Mikel Landa")
    for(ciclista in clasificacion){//Bucle for para recorrer el mapa
        println(ciclista.value)
        println(ciclista.key)
    }

    for(i in 1..clasificacion.count()){//For desde 1 hasta la cantidad de elementos en el mapa
        println(i)
    }
}
```

```
SeppKuss
1
Kiam Auisp
2
Mikel Landa
3
1
2
3
```

Sentencia IF

```
fun main() {
    val clasificacion: Map<Int, String> = mapOf(1 to "SeppKuss", 2 to "Kiam Auisp", 3 to "Mikel Landa")
    for(ciclista in clasificacion){//Bucle for para recorrer el mapa
        if(ciclista.key == 1){//Concicional IF
            println("El ciclista ${ciclista.value} ha ganado")
        }else{
            println("El ciclista ${ciclista.value} ha quedado en la posición ${ciclista.key}")
        }
        println(ciclista.value)
        println(ciclista.key)
    }

    for(i in 1..clasificacion.count()){//For desde 1 hasta la cantidad de elementos en el mapa
        println(i)
    }
}
```

```
El ciclista SeppKuss ha ganado
SeppKuss
1
El ciclista Kiam Auisp ha quedado en la posición 2
Kiam Auisp
2
El ciclista Mikel Landa ha quedado en la posición 3
Mikel Landa
3
1
```

When

Equivale a un switch de C

```
fun main() {
    val clasificacion: Map<Int, String> = mapOf(1 to "SeppKuss", 2 to "Kiam Auisp", 3 to "Mikel Landa")
    for(ciclista in clasificacion){//Bucle for para recorrer el mapa
        if(ciclista.key == 1){//Concicional IF
            println("El ciclista ${ciclista.value} ha ganado")
        }else{
            println("El ciclista ${ciclista.value} ha quedado en la posición ${ciclista.key}")
        }
        println(ciclista.value)
        println(ciclista.key)

        var mensaje = when{//equivalente al Switch
        ciclista.key == 1 -> "${ciclista.value} ha ganado la carrera"
        ciclista.key > 1 && ciclista.key <=3 -> "${ciclista.value} ha subido al podio"
        else -> "${ciclista.value} ha quedado en la posición ${ciclista.key}"
        }
    println(mensaje)

    }
```

```
SeppKuss
1
SeppKuss ha ganado la carrera
El ciclista Kiam Auisp ha quedado en la posición 2
Kiam Auisp
2
Kiam Auisp ha subido al podio
El ciclista Mikel Landa ha quedado en la posición 3
Mikel Landa
3
Mikel Landa ha subido al podio
```

Funciones

```kotlin
import kotlin.random.Random

fun correrCarrera():Int{
    return Random.nextInt(1,190)
}

fun mostrarClasificacion( nombre:String, dorsal:Int, posicion:Int):Unit{
    println("El ciclista $nombre (dorsal n=$dorsal) ha terminado en la posicion $posicion")
}

fun main():Unit {
    var posicion = correrCarrera()
    println(posicion)
    mostrarClasificacion("Daniel Valero", 11, posicion)
}
```

```
116
El ciclista Daniel Valero (dorsal n=11) ha terminado en la posicion 116
```

Clases

Las clases pueden ser Public, Private, Protected y internal:

- public: Todos los ven
- open: para que pueda tener subclases
- Private: Solo la clase lo puede ver
- Protected: solo las subclases la ven
- Internal: solo se puede ver si está dentro del mismo paquete

```kotlin
class Persona(val nombre:String, val edad:Int, val equipo:String){//Declaramos la clase persona y sus atributos

}

fun main() {
    var persona1 = Persona("patata", 33, "Universidad de cámaras")//Creamos objeto de clase Persona
    println(persona1.nombre)//Mostramos atributo nombre
    println(persona1.edad)
    println(persona1.equipo)

}
```

```
patata
33
Universidad de cámaras
```

Herencia

```
open class Persona(val nombre:String, val edad:Int, val equipo:String){//Declaramos la clase persona y sus atributos
    open fun ParticipaCarrera(){//método que pueden utilizar los hijos
        println("Mucho texto")
    }
}

class Ciclista(nombre:String, edad:Int, equipo: String):Persona(nombre, edad, equipo){//Ciclista hereda de Persona
    val sprint: Int = 55//Nuevo atributo específico para ciclista
}

fun main() {
    var persona1 = Persona("patata", 33, "Universidad de cámaras")//Creamos objeto de clase Persona
    println(persona1.nombre)//Mostramos atributo nombre

    var ciclista1 = Ciclista("Daniel", 88, "Universidad de vacas")
    println("El ${ciclista1.nombre} tiene un valor de sprint de ${ciclista1.sprint}")

}
```

```
patata
El Daniel tiene un valor de sprint de 55
```

Abstracción

```kotlin
abstract class Animal(){
    abstract fun sonido()
}

class Gato(val nombre:String, val edad:Int):Animal(){
    override fun sonido(){//sobreescribimos el método
        println("El gato hace meow")
    }
}

class Perro(val nombre:String, val edad:Int):Animal(){
    override fun sonido(){
        println("Cuack")
    }
}

fun main() {
    var gato = Gato("Garfield", 3)
    gato.sonido()

    var perro = Perro("LePerro", 1)
    perro.sonido()
}
```

```
El gato hace meow
Cuack
```

Clases Anidadas e Internas

```
class Ciclista{
    var sprint: Int= 55
    class MedicinaNested{//Clase Anidada
        fun dopar(): Double{//No se puede acceder a atributos de fuera desde clase anidadad
            var sprint = 50
            return sprint*1.25
        }
    }

    inner class MedicinaInterna{//clase Interna
        fun dopar(): Double{//Puede acceder a atributos del padre
            return sprint*1.25
        }
    }
}
fun main() {
    var ciclista = Ciclista.MedicinaNested()
    var dopaje = ciclista.dopar()
    println(dopaje)
    var ciclista2 = Ciclista().MedicinaInterna()
    dopaje = ciclista2.dopar()
    println(dopaje)
}
```

```
62.5
68.75
```

Clase Data

```
data class Ciclista(val  nombre:String, val edad:Int, val equipo:String){

}
fun main() {
    var ciclista = Ciclista("Dani", 33, "UCAM")
    var ciclista2 = Ciclista("Juan",22, "UMU")
    var ciclista3 = ciclista.copy()//Copiamos el valor de ciclista a ciclista 3

    println(ciclista.toString())//Pasa dato a cadena de texto
    println(ciclista.equals(ciclista2))//Comparador
    println(ciclista.equals(ciclista3))

    val (nombre, edad , equipo) = ciclista//Creamos una tupla
    println("$nombre $edad $equipo")
}
```

```
Ciclista(nombre=Dani, edad=33, equipo=UCAM)
false
true
Dani 33 UCAM
```

Interface y Herencia múltiple

```
open class Persona(val nombre:String, val edad:Int){//case padre
    open fun participa(){
        println("la persona $nombre esta participando")
    }
}
interface ParticiparCarrera{//Interfaz
    fun sprintar()
    fun atacar(){
        println("El ciclista usa impactrueno")
    }
}
class Ciclista(nombre:String, edad:Int):Persona(nombre,edad), ParticiparCarrera{//herencia multiple de Clase e Interfaz
    override fun sprintar(){
        println("El ciclista $nombre sprinta")
    }
}
fun main() {
    var ciclista = Ciclista("patata", 11)
    ciclista.participa()
    ciclista.sprintar()
    ciclista.atacar()
}
```

```
la persona patata esta participando
El ciclista patata sprinta
El ciclista usa impactrueno
```

Funciones Lambda

```
fun main() {//Funciones Lambda
    val mult = fun (x:Int, y:Int) = x*y
    val sum = fun (x:Int, y: Int) = x+y
    fun funcionLambda(x:Int, y:Int, miFuncion:(Int,Int)->Int):Int{//Metemos 2 variables y una función
        return miFuncion(x,y)//devolvemos la función recibiendo como entradas las variables
    }

    var res = funcionLambda(4,5,mult)//Funcion lambda en uso
    println(res)
    res = funcionLambda(4,5,sum)
    println(res)

}
```

```
20
9
```