UNIVERSIDAD DE LIMA | 60 años

Fundamentos de Kotlin

Programación Móvil – ULima (hquintan@ulima.edu.pe)

Kotlin

- Es un lenguaje que se compila a bytecode como Java, lo que nos da 100% de interoperabilidad.

- Tiene una librería de ejecución bastante compacta (contribuye poco al tamaño del apk).

REPL

- Kotlin cuenta con un shell interactivo para poder ejecutar sentencias de Kotlin y poder probar nuestro código.

```
fun printHola(){
  println("Hola Mundo")
}
printHola()
```

```
1+1

40-5

40/2

1/2

1.0 / 2.0
```

UNIVERSIDAD DE LIMA

Operaciones como métodos

- Todo variable en Kotlin puede tratarse como un objeto.
- Si bien es cierto, los identificadores pueden tratarse también como objetos, son variables primitivas.
- A no ser que se le coloque un wrapper sobre el identificador. (boxed mode).Tener cuidado de malutilizar los boxed variables.

```
val num = 6
num.times( 6 )
num.div( 2 )
num.plus( 4 )
num.minus( 6 )
```

UNIVERSIDAD DE LIMA

Definición de tipos

- Es posible definer un tipo de dato explícito para una variable.
- Se utilizan ":" para definir el tipo de dato dela variable

```
val boxedNum : Number = 5
num.toLong()
```

Tipos de variables

- No modificables. El valor de estas no puede modificarse (también llamadas **inmutables**).

```
val numero = 3
numero = 4
```

- Modificables. Su valor puede modificarse (**mutables**).

```
var numero = 3
numero = 4
```

UNIVERSIDAD DE LIMA

Los tipos de las variables pueden ser inferidos

- El tipo de las variables es inferido en caso de que no esté explícitamente definido.
- En la compilación, ya se le asigna el tipo de dato dependiendo del contexto (no significa que su tipo de datos es cambiable).

```
var num = 3
num = "Hola"
```

ERROR

UNIVERSIDAD DE LIMA

Casting

- Se permite el *explicit casting* pero no el *implicit casting*.

```
var num : Int
num = 10.0 / 2.0
```

ERROR

```
var num : Int
num = (10.0 / 2.0).toInt()
```

UNIVERSIDAD DE LIMA

Formateo de números

- Es posible escribir ciertos números de una manera que sean más entendibles por el programador (no tiene impacto en la compilación y ejecución).

```
val num = 1_000_000
val hexNum = 0xFE_EC_DE_5E
val bitNum = 0b1101_1100_0011_0001
```

Null types

- Por defecto, una variable no puede ser nula. Si se desea que sea deberá de agregarse al tipo de la variable el signo "?".

```
var num : Number = null
var num : Number? = null
```

num no puede ser nulo, por eso da error.

num es de tipo Number?, también llamado Optional. Esto significa que puede ser nulo.

UNIVERSIDAD DE LIMA

Optionals

- Sirven para delegar la responsabilidad del manejo de nulos al programador y no apoyarse en las excepciones.

```
var num : Number? = funPuedeDevolverNulo()
//…
if (num == null){
  // Codigo de manejo del caso que sea nulo
}else{
  num.toInt()
}
```

- Se puede obviar la advertencia y decidir utilizar la variable que puede ser nula (no aconsejable).

```
num!!.toInt()
```

En caso que num sea nulo, entonces se generará un NullPointException

UNIVERSIDAD DE LIMA

Otra manera de tratar Optionals

- Para una verificación más sencilla de si una variable toma el valor de nulo, se puede utilizar la siguiente forma:

```
val numEntero = num?.toInt() ?: 0
```

- num?.length daría null, pero se utiliza el operador llamado elvis operator por lo que se asignaría a numEntero el valor de 0.

UNIVERSIDAD DE LIMA

Orientación a objetos

Clases

- Se puede acortar la inicialización de una clase y de sus propiedades.
- Tener cuidado si es que se tienen constructores secundarios.
- También puede definir un bloque de inicialización (init)

```
class Person(val firstName: String, val lastName: String, var age: Int) {
    //...
}
```

```
class Person(val name: String) {
    constructor(name: String, parent: Person) : this(name) {
        parent.children.add(this)
    }
}
```

```
class Person(val name: String) {
    init {
        // Tareas de inicializacion
    }
}
```

https://kotlinlang.org/docs/reference/classes.html

UNIVERSIDAD DE LIMA

Herencia

- Tomar en cuenta que se debe declarar la clase padre como open.

```
open class Base(p: Int)

class Derived(p: Int) : Base(p)
```

- Todas las clases en Kotlin heredan de una clase base llamada Any.
- En caso de que haya constructors secundarios, se puede utilizar el keyword super.

```
class MyView : View {
    constructor(ctx: Context) : super(ctx)

    constructor(ctx: Context, attrs: AttributeSet) : super(ctx, attrs)
}
```

UNIVERSIDAD DE LIMA

Sobrescritura

- Puedes sobrescribir métodos y propiedades (atributos)

```
open class Base {
    open fun v() { ... }
    fun nv() { ... }
}
class Derived() : Base() {
    override fun v() { ... }
}
```

```
open class Foo {
    open val x: Int get() {
... }
}

class Bar1 : Foo() {
    override val x: Int = ...
}
```

Tiene que ser explícito!

- En caso de querer que un método no sea sobreescribible, deberá de marcarse en la clase Base.

```
open class AnotherDerived() : Base() {
    final override fun v() { ... }
}
```

UNIVERSIDAD DE LIMA

Clases Data

- Al declarar una clase como data significa que queremos que sea tratada como un POJO (Kotlin genera ciertos métodos como por ejemplo el *equals*.

```
data class User(val name: String, val age: Int)
```

- En caso que declaremos variables dentro del cuerpo de la clase, estas no entran dentro de los métodos generados.

```
data class Person(val name: String) {
    var age: Int = 0
}
```

UNIVERSIDAD DE LIMA

UNIVERSIDAD DE LIMA

Funciones y lambdas

Tipos de Funciones

- Se definen siguiendo la siguiente sintaxis.

```
val f : (Number) -> String
```

- En este caso se esta definiendo una variable llamada f que es de tipo de una función que recibe un Number y devuelve un String.

Funciones de alto orden

- Declarando implementación de las funciones:

Lamba

```
{ a, b -> a + b }
```

Función anónima

```
fun(s: String): Int { return s.toIntOrNull() ?: 0 }
```

UNIVERSIDAD DE LIMA

Companion objects

- La manera como Kotlin maneja las propiedades de clase (estáticos).
- A diferencia de Java, un class no es un objeto directamente, si no se le tiene que especificar un **companion object**.

```
class Persona(val nombre : String, val edad : Int) {
    init {
        Persona.cantidadPersonas++
    }
    companion object {
        private var cantidadPersonas : Int = 0

        fun getCantidadPersonas() : Int{
            return cantidadPersonas
        }
    }
}
```

```
val persona1  = Persona("Pepe", 20)
val persona2 = Persona("Pedro", 20)
println(Persona.getCantidadPersonas())
```

UNIVERSIDAD DE LIMA

Estructuras de Datos

- Listas:
  - List: No puede ser modificada luego de creada.
  - MutableList: Puede ser modificada luego de creada.
  - Pueden utilizar la función listOf para crearlas.

- HashMap: Permite guardar una lista de tuplas llave, valor.
  - Puede utilizar la función hashMapOf para crearlas.

```
val tabla = hashMapOf("nombre" to "Pepe", "edad" to 20)
tabla["nombre"]
```

```
class User(val map: Map<String, Any?>) {
    val name: String by map
    val age: Int     by map
}

val user = User(
    hashMapOf(
        "name" to "John Doe",
        "age"  to 25
    )
)
```

UNIVERSIDAD DE LIMA

Referencias

- https://kotlinlang.org/docs/reference/
- https://play.kotlinlang.org/koans/overview