KOTLIN BASICS

Yeratzy Espinoza

Kotlin

Un lenguaje de programación moderno que ayuda a los desarrolladores a ser más productivos.

Expresivo y conciso

Código más seguro

Interoperable

Simultaneidad estructurada

VARIABLES

Inferencia de tipos

- El compilador puede inferir el tipo.
- Se puede declarar el tipo explícitamente si es necesario.

Variables mutables e inmutables

- La inmutabilidad no es forzada, pero se recomienda.

Kotlin es un lenguaje de tipado estático. El tipo se resuelve en tiempo de compilación y nunca cambia.

Mutable and immutable variables

- Mutable

```
var contador = 10
```

- Inmutable

```
val nombre = “Carmen”
```

DATA TYPES

Types

Integer

Long
Int
Short
Byte

Non-Integer

Double
Float
Char
Boolean

Type casting

```
val a: Int = 9
val b: Byte = a
println(b)
```

⇒ error: type mismatch: inferred type is Int but Byte was expected

```
val a: Int = 9
println(a.toByte())
```

⇒ 9

Numeric operator methods

Kotlin mantiene los números como primitivos, pero le permite llamar a métodos sobre números como si fueran objetos.

```
2.times(3)
⇒ Kotlin.Int = 6
```

```
2.4.div(2)
⇒ Kotlin.Double = 1.2
```

```
3.5.plus(4)
⇒ Kotlin.Double = 7.5
```

Strings

$variable –> Se llama **interpolación variable**

Concatenación

```
val numFrutas = 4
val numVerduras = 3

println("Tengo $numFrutas frutas" + "y $numVerduras verduras")
```

⇒ Tengo 4 frutas y 3 verduras

Strings templates

La expresión template comienza con un signo de pesos ($) y puede ser un valor simple:

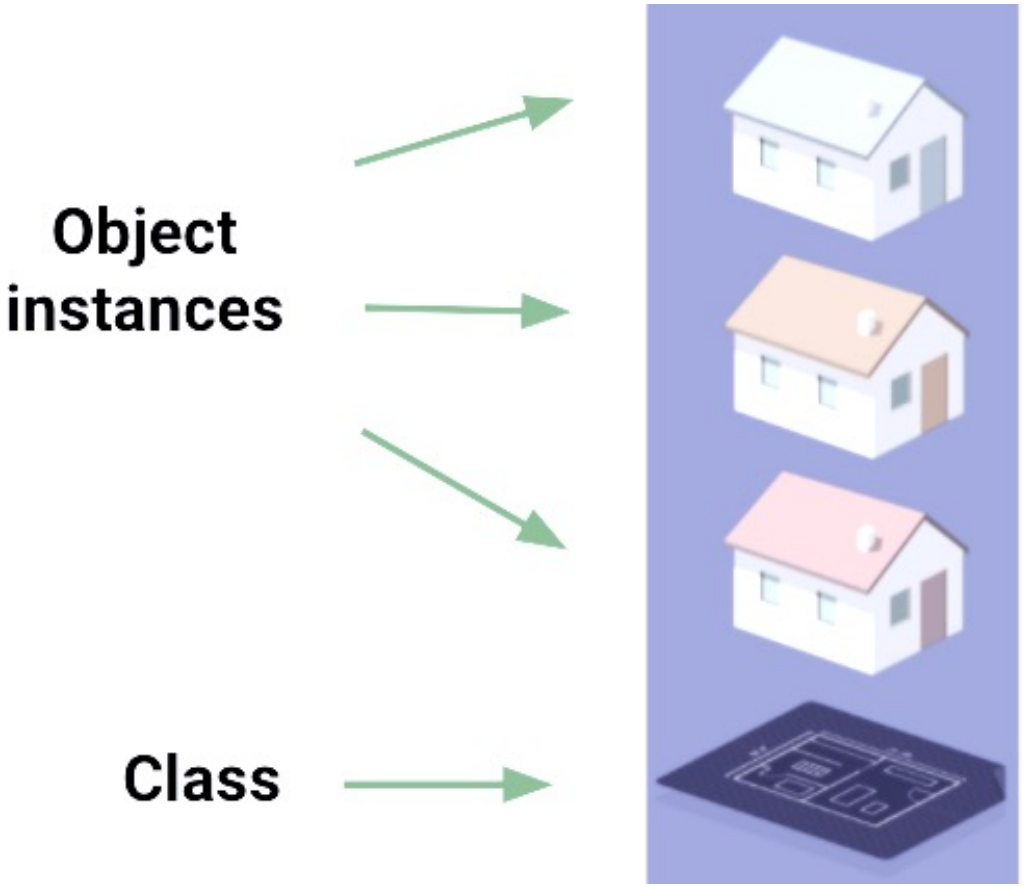
```
val i = 10
println("i = $i")
⇒ i = 10
```

O como una expresión dentro de llaves:

```
val x = "hola"
println("$x.length es ${x.length}")
⇒ hola.length es4
```

CLASSES

- Las clases son planos de objetos
- Las clases definen métodos que operan en sus instancias de objetos



Define and use a class

```
class House {
  val color: String = "white"
  val numberOfWindows: Int = 2
  val isForSale: Boolean = false

  fun updateColor(newColor: String){...}
  ...
}
```

Object Instance

```
val myHouse = House()
println(myHouse)
```

Constructors

Cuando se define un constructor en el encabezado de la clase, puede contener:

- Sin parámetros

  class A

- Parámetros
  - No marcado con var o val → la copia existe solo dentro del alcance del constructor

    class B(x: Int)

  - Marcados con var o val → la copia existe en todas las instancias de la clase
    - class B(val y: Int)

NULL SAFETY

Null safety

- Las variables no pueden ser nulas por default.
- Se puede asignar explícitamente una variable nula utilizando el operador de llamada segura.
- Permite excepciones null-pointer utilizando el operador !!
- Se pueden probar los nulos utilizando el elvis(?:) operator

Las variables no pueden ser nulas

- Las variables nulas no están permitidas por defecto.

Declara una variable Int y asignale null:

```
var numberOfSomething: Int = null
```

⇒ error: null can not be a value of a non-null type Int

Safe call operator

- El operador de llamada segura (?), después del tipo indica que la variable puede ser nula.

Declara un Int? as nullable

var numberOfSomething : Int? = null

En general, no establezca una variable en nula, ya que puede tener consecuencias no deseadas.

Testing for null

- Comprobar si la variable numberOfSomething no es nula. Luego decrementa esa variable.

```
var numberOfSomething = 6
  if (numberOfSomething != null) {
       numberOfSomething = numberOfSomething.dec()
  }
```

- En Kotlin se puede escribir usando el operador de llamada segura.

```
var numberOfSomething = 6
   numberOfSomething = numberOfSomething?.dec()
```

The !! operator

- Si está seguro de que una variable no será nula, use !! para forzar la variable a un tipo no nulo.
- Entonces puedes llamar a métodos/propiedades en él.

```
val len = s!!.length
        => Lanza NullPointerException si es nulo
```

Advertencia: !! arrojará una excepción, solo debe usarse cuando sea excepcional mantener un valor nulo.

Elvis operator

- Realiza pruebas nulas con el operador ?:

```
numberOfSomething = numberOfSomething?.dec() ?: 0
```

COMPANION OBJECTS

Companion objects

- Permite que todas las instancias de una clase compartan una única instancia de un conjunto de variables o funciones.
- Usar palabra clave companion.
- Referenciado a través de ClassName.PropertyOrFunction.

Companion object Ejemplo

```
class PhysicsSystem {
    companion object WorldConstants {
        val gravity = 9.8
        val unit = "metric"
        fun computeForce(mass: Double, accel: Double): Double {
            return mass * accel
        }
    }
}
println(PhysicsSystem.WorldConstants.gravity)
println(PhysicsSystem.WorldConstants.computeForce(10.0, 10.0))
=> 9.8100.0
```