UD07 – INTRODUCCIÓN A KOTLIN

Contenido

[1 Lenguaje de programación Kotlin 3](#_Toc191379859)

[1.1 Características del lenguaje 3](#_Toc191379860)

[1.2 Programa *Hello World* en Kotlin 3](#_Toc191379861)

[1.2.1 Programa "Hola Mundo" en Kotlin 4](#_Toc191379865)

[1.2.2 Espacios de nombres en Kotlin 4](#_Toc191379866)

[1.2.3 Compilación y ejecución 4](#_Toc191379867)

[1.3 Entrada y salida de datos por consola 5](#_Toc191379868)

[2 Estructura de un programa en Kotlin 5](#_Toc191379869)

[2.1 Organización de un proyecto en Kotlin 6](#_Toc191379870)

[2.1.1 Creación de un módulo de biblioteca en Kotlin 6](#_Toc191379873)

[2.1.2 Creación de una aplicación de consola en Kotlin (Equivalente a AcumuladorApp.exe) 6](#_Toc191379874)

[2.1.3 Manejo de múltiples archivos en Kotlin 7](#_Toc191379875)

[3 Un vistazo a lenguaje Kotlin 8](#_Toc191379876)

[3.1 Comentarios 8](#_Toc191379877)

[3.2 Tipos y variables 8](#_Toc191379878)

[3.2.1 Tipos de valor en Kotlin 9](#_Toc191379882)

[3.2.2 Tipos de referencia en Kotlin 9](#_Toc191379883)

[3.3 Expresiones 10](#_Toc191379884)

[3.3.1 Precedencia y Asociatividad 10](#_Toc191379886)

[3.3.2 Operadores aritméticos 10](#_Toc191379887)

[3.3.3 Operadores lógicos 11](#_Toc191379888)

[3.3.4 Operadores de comparación 11](#_Toc191379889)

[3.4 Instrucciones 12](#_Toc191379890)

[3.4.1 Bloques e Instrucciones de Declaración 12](#_Toc191379892)

[3.4.2 Instrucciones de Expresión 12](#_Toc191379893)

[3.4.3 Instrucciones de Selección (if, when) 13](#_Toc191379894)

[3.4.4 Instrucciones de Iteración (while, do-while, for) 14](#_Toc191379895)

[3.4.5 Instrucciones de Salto (break, continue, return) 15](#_Toc191379896)

[3.4.6 Manejo de Excepciones (try-catch-finally) 16](#_Toc191379897)

[4 Clases y objetos 17](#_Toc191379898)

[4.1 Miembros de una clase 18](#_Toc191379899)

[4.2 Propiedades 21](#_Toc191379900)

[4.3 Constructores 22](#_Toc191379901)

[4.4 Destructores 22](#_Toc191379902)

[4.5 Modificadores de acceso 23](#_Toc191379903)

[5 Tipos de datos integrados 23](#_Toc191379904)

[5.1 Tipos de enteros 24](#_Toc191379905)

[5.2 Tipo de dato real: simple y doble precisión, cómo mostrar en pantalla. 25](#_Toc191379906)

[5.3 Conversiones de tipo. 26](#_Toc191379907)

[5.4 Tipo char 27](#_Toc191379908)

[5.5 Tipo booleano 28](#_Toc191379909)

[5.6 Cadenas de caracteres 28](#_Toc191379910)

[8.6.1 Conversiones entre tipos numéricos y cadenas 32](#_Toc191379922)

[8.6.2 Clase StringBuilder 38](#_Toc191379923)

[6 Declaración de variables 40](#_Toc191379924)

[7 Operadores 44](#_Toc191379925)

[7.1 Operadores aritméticos 44](#_Toc191379926)

[10.1.1 Arithmetic Overflow 46](#_Toc191379930)

[10.1.2 Prioridad de operadores aritméticos 47](#_Toc191379931)

[7.2 Operadores lógicos booleanos 47](#_Toc191379932)

[7.3 Operadores de igualdad 49](#_Toc191379935)

[7.4 Operadores de orden 50](#_Toc191379936)

[7.5 Operadores a nivel de bit 51](#_Toc191379937)

[8 Tipos de enumeración. Enum. 54](#_Toc191379938)

[8.1 Tipos de enumeración como bit flags 57](#_Toc191379939)

[9 Tipos de valor que aceptan valores null 59](#_Toc191379940)

[10 Structs 60](#_Toc191379941)

[11 Arrays 63](#_Toc191379942)

[11.1 Arrays multidimensionales 65](#_Toc191379943)

[11.2 Introducción a las listas en C# 70](#_Toc191379944)

[12 Funciones 71](#_Toc191379945)

[13 Tuplas 75](#_Toc191379946)

[14 Tipos de fecha y hora 77](#_Toc191379947)

# **Lenguaje de programación Kotlin**

Kotlin es un lenguaje de programación moderno, conciso y seguro desarrollado por JetBrains. Se diseñó para ser completamente interoperable con Java, permitiendo a los desarrolladores aprovechar las bibliotecas y herramientas existentes. Su sintaxis clara y expresiva lo hace fácil de aprender para quienes tienen experiencia en lenguajes como Java, C#, Swift o JavaScript.

Kotlin se destaca por su enfoque en la seguridad de tipos, eliminando errores comunes como los NullPointerException, además de ofrecer potentes características como corutinas para programación asíncrona y una gran compatibilidad con el desarrollo de aplicaciones Android, backend y multiplataforma.

## **Características del lenguaje**

Kotlin es un lenguaje de programación moderno y orientado a objetos que también admite paradigmas funcionales. Sus características facilitan la construcción de aplicaciones seguras, eficientes y mantenibles.

* Recolección de basura (Garbage Collection): Al igual que Java, Kotlin se ejecuta en la JVM y utiliza un recolector de basura para liberar automáticamente la memoria ocupada por objetos que ya no son accesibles, evitando problemas de administración manual de memoria.
* Manejo de excepciones: Kotlin proporciona un sistema estructurado para la captura y manejo de errores mediante try-catch-finally, asegurando una recuperación efectiva ante fallos en la ejecución.
* Seguridad de tipos: Kotlin minimiza errores comunes con su sistema de tipos seguro, evitando problemas como el acceso a variables no inicializadas, desbordamiento de arrays o conversiones de tipos inseguras.
* Sistema de tipos unificado: Aunque Kotlin diferencia entre tipos primitivos (Int, Double, etc.) y tipos de referencia, el compilador optimiza el uso de memoria, reduciendo la sobrecarga innecesaria. Además, Kotlin permite la creación de clases de datos, tipos sellados y tipos nulos seguros para mayor flexibilidad y seguridad.
* Soporte para programación funcional: Kotlin incorpora características como funciones de orden superior, expresiones lambda e inmutabilidad para mejorar la expresividad del código y facilitar el desarrollo de software robusto.

## **Programa *Hello World* en Kotlin**

Para crear un proyecto de consola en Kotlin utilizando IntelliJ IDEA, sigue estos pasos:

1. Abrir IntelliJ IDEA y seleccionar "Nuevo Proyecto".
2. Elegir la plantilla "Aplicación Kotlin/JVM".
3. Asignar el nombre del proyecto como Hello.
4. Seleccionar el JDK (Java Development Kit) compatible, como JDK 17 o superior.
5. Pulsar "Crear".

Normalmente, los archivos de código fuente de Kotlin tienen la extensión .kt.



### Programa "Hola Mundo" en Kotlin

Podemos escribir el código directamente en el archivo Main.kt y ejecutarlo. En Kotlin, println es la función que nos permite escribir en la consola.

fun main() {

println("Hola Mundo")

}

### Espacios de nombres en Kotlin

Para importar el espacio de nombres, en Kotlin utilizamos import para importar paquetes específicos. Por ejemplo, si queremos trabajar con entradas de usuario, podemos importar:

import java.util.Scanner

Kotlin organiza sus bibliotecas de manera similar a los paquetes en Java, permitiendo la reutilización y estructuración del código.

### Compilación y ejecución

Cuando compilamos nuestro proyecto en IntelliJ IDEA o con Kotlin Compiler, se generará un archivo .jar ejecutable.

Si usamos la línea de comandos, podemos compilar y ejecutar con:

kotlinc Main.kt -include-runtime -d Hello.jar

java -jar Hello.jar

## **Entrada y salida de datos por consola**

En Kotlin, la entrada y salida estándar se maneja con funciones de la biblioteca estándar, sin necesidad de una clase específica como Console en C#. A continuación, te muestro los métodos equivalentes en Kotlin:

Métodos de salida

|  |  |
| --- | --- |
| Kotlin | Descripción |
| print(valor) | Escribe el valor sin salto de línea. |
| println(valor) | Escribe el valor seguido de un salto de línea. |

Métodos de entrada

|  |  |
| --- | --- |
| Kotlin | Descripción |
| System.`in.read()` | Lee un carácter de la entrada estándar. |
| readLine() | Lee una línea completa de la entrada estándar. |

Ejemplo en Kotlin

fun main() {

print("Introduce tu nombre: ") // print() no agrega salto de línea

val nombre = readLine() // Captura la entrada del usuario

println("Hola, $nombre") // println() agrega un salto de línea

}

# **Estructura de un programa en Kotlin**

En Kotlin, los principales conceptos organizativos incluyen paquetes, clases, funciones y módulos.

* Los programas de Kotlin constan de archivos fuente con la extensión .kt.
* Los paquetes organizan las clases y funciones de manera jerárquica, similar a los espacios de nombres en C#.
* Los tipos en Kotlin incluyen clases, interfaces, objetos y enumeraciones.
* Los módulos agrupan varios archivos fuente y pueden compilarse en JARs o bibliotecas compartidas.

## Organización de un proyecto en Kotlin



### Creación de un módulo de biblioteca en Kotlin

En IntelliJ IDEA, sigue estos pasos para crear una biblioteca reutilizable:

* Crear un nuevo proyecto en IntelliJ IDEA.
* Seleccionar "Aplicación Kotlin/JVM".
* Agregar un nuevo módulo de tipo "Biblioteca", nombrándolo Acumulador.
* Crear una clase Valor.kt dentro del paquete com.ejemplo.acumulador.

package com.ejemplo.acumulador

class Valor {

var valor: Int = 0

fun acumular(aumento: Int) {

valor += aumento

}

}

### Creación de una aplicación de consola en Kotlin (Equivalente a AcumuladorApp.exe)

* Crear un nuevo proyecto llamado AcumuladorApp.
* Agregar una dependencia al JAR Acumulador.jar generado previamente.
* Escribir el siguiente código en Main.kt:

package com.ejemplo.acumuladorapp

import com.ejemplo.acumulador.Valor

fun main() {

val miValor = Valor()

miValor.acumular(10)

println("El valor acumulado es: ${miValor.valor}")

}

### **Manejo de múltiples archivos en Kotlin**

Ejemplo de dividir la clase Valor en dos archivos:

* Archivo Valor.kt

package com.ejemplo.acumulador

class Valor {

var valor: Int = 0

}

* Archivo ValorMetodos.kt

package com.ejemplo.acumulador

fun Valor.acumular(aumento: Int) {

this.valor += aumento

}

Al compilar, Valor tendrá acceso a acumular(), aunque esté definido en otro archivo.

# **Un vistazo a lenguaje Kotlin**

## Comentarios

En Kotlin, los comentarios siguen la misma sintaxis que en C# y Java:

// Comentario en una línea

/\* Comentario en

\* varias líneas

\*/

También se pueden usar **comentarios de documentación** con /\*\* ... \*/, similares a /// en C#.

/\*\*

\* Esto es un comentario de documentación en Kotlin

\*/

fun ejemplo() {}

## Tipos y variables

Kotlin también distingue entre **tipos de valor (primitivos)** y **tipos de referencia (objetos)**. Sin embargo, a diferencia de C#, los tipos primitivos en Kotlin son **representados internamente como tipos primitivos de JVM** cuando es posible, lo que mejora la eficiencia.



### Tipos de valor en Kotlin

|  |  |
| --- | --- |
| Categoría | Lenguaje Kotlin |
| Enteros con signo | Byte, Short, Int, Long |
| Enteros sin signo | UByte, UShort, UInt, ULong (desde Kotlin 1.3) |
| Caracteres Unicode | Char |
| Punto flotante binario IEEE | Float, Double |
| Punto flotante decimal de alta precisión | No tiene equivalente directo |
| Booleanos | Boolean |

Ejemplo de declaración de variables en Kotlin:

val numero: Int = 10 // Variable inmutable

var texto: String = "Hola" // Variable mutable

### Tipos de referencia en Kotlin

|  |  |
| --- | --- |
| Categoría | Lenguaje Kotlin |
| Clase base de todos los tipos | Any |
| Cadenas Unicode Cadenas Unicode | String |
| Clases definidas por el usuario | class MiClase {} |
| Interfaces | interface MiInterfaz {} |
| Arrays | Array<Int> o IntArray |

Ejemplo de clases en Kotlin:

class Persona(val nombre: String, var edad: Int)

Ejemplo de una interfaz en Kotlin:

interface Animal {

fun hacerSonido()

}

class Perro : Animal {

override fun hacerSonido() {

println("Guau!")

}

}

## Expresiones

Las expresiones en Kotlin también se construyen con operandos y operadores. Los operadores indican qué operaciones se aplican a los operandos, y la precedencia y asociatividad determinan el orden de evaluación.



### **Precedencia y Asociatividad**

* Precedencia de operadores en Kotlin es similar a la de C#.
* Asociatividad en Kotlin sigue estas reglas:

Los operadores binarios (como +, -, \*, /) son asociativos a la izquierda:

val resultado = 10 - 5 - 2 // Se evalúa como (10 - 5) - 2 = 3

Los operadores de asignación (=) y el operador ternario ?: son asociativos a la derecha:

val x: Int = 5

val y: Int = 10

val z: Int = x + y // Se evalúa como (x + y)

### Operadores aritméticos

Estos operadores funcionan igual que en Java:

|  |  |  |
| --- | --- | --- |
| Operador | Descripción | Ejemplo |
| + | Suma | val suma = 5 + 3 → 8 |
| - | Resta | val resta = 5 - 3 → 2 |
| \* | Multiplicación | val producto = 5 \* 3 → 15 |
| / | División | val division = 5 / 2 → 2 (división entera) |
| % | Módulo (resto) | val resto = 5 % 2 → 1 |

En Kotlin, la división entre Int descarta la parte decimal. Para obtener un resultado decimal, uno de los operandos debe ser Double o Float:

val resultado = 5 / 2.0 // Resultado: 2.5

### Operadores lógicos

Se usan para trabajar con valores booleanos (true o false).

|  |  |  |
| --- | --- | --- |
| Operador | Descripción | Ejemplo |
| ! | Negación lógica (NOT) | val esFalso = !true → false |
| && | AND lógico | val resultado = true && false → false |

### Operadores de comparación

val a = "Hola"

val b = "Hola"

println(a == b) // true (compara valores)

println(a === b) // false (compara referencias)

|  |  |  |
| --- | --- | --- |
| Operador | Descripción | Ejemplo |
| == | Igual a | val esIgual = (5 == 5) → true |
| != | No igual a | val esDistinto = (5 != 3) → true |
| < | Menor que | val menor = (3 < 5) → true |
| <= | Menor o igual que | val menorIgual = (3 <= 3) → true |
| > | Mayor que | val mayor = (5 > 3) → true |
| >= | Mayor o igual que | val mayorIgual = (5 >= 5) → true |

## Instrucciones

Kotlin tiene muchas similitudes con Java en cuanto a las instrucciones de control de flujo, pero con una sintaxis más concisa y expresiva. Veamos las principales diferencias y similitudes.



### Bloques e Instrucciones de Declaración

* Bloques en Kotlin se definen con { }.
* Declaraciones de variables: En Kotlin, usamos val (inmutable) y var (mutable) en lugar de const y tipos explícitos.

Ejemplo en Kotlin:

fun declarations() {

var a: Int

var b = 2

val c = 3 // `val` es como `const` en C#

a = 1

println(a + b + c)

}

Diferencias clave:

val es inmutable pero no una constante en tiempo de compilación.

Para valores constantes, se usa const val (solo dentro de object o companion object).

const val PI = 3.1415927

### Instrucciones de Expresión

En Kotlin, cualquier expresión puede convertirse en una instrucción:

fun expressions() {

var i: Int

i = 123

println(i)

i++

println(i)

}

Kotlin no requiere punto y coma ; al final de cada línea (a diferencia de Java).

Las acciones de un programa se expresan mediante *instrucciones*. C# admite varios tipos de instrucciones diferentes, varias de las cuales se definen en términos de instrucciones insertadas.

Un *bloque* permite que se escriban varias instrucciones en contextos donde se permite una única instrucción. Un bloque se compone de una lista de instrucciones escritas entre los delimitadores **{  }**.

### Instrucciones de Selección (if, when)

* IF:

fun ifStatement(args: Array<String>) {

if (args.isEmpty()) {

println("No arguments")

} else {

println("One or more arguments")

}

}

* switch en Kotlin → when

Kotlin reemplaza switch por when, que es más flexible:

when no necesita break porque se detiene automáticamente después de una coincidencia.

fun switchStatement(args: Array<String>) {

when (args.size) {

0 -> println("No arguments")

1 -> println("One argument")

else -> println("${args.size} arguments")

}

}

### Instrucciones de Iteración (while, do-while, for)

* while en Kotlin

fun whileStatement(args: Array<String>) {

var i = 0

while (i < args.size) {

println(args[i])

i++

}

}

* do-while en Kotlin

fun doStatement() {

var s: String?

do {

s = readlnOrNull()

println(s)

} while (!s.isNullOrEmpty())

}

* for en Kotlin

En Kotlin, el for estándar usa rangos en lugar de índices:

fun forStatement(args: Array<String>) {

for (i in args.indices) {

println(args[i])

}

}

También se puede escribir con forEach:

args.forEach { println(it) }

### Instrucciones de Salto (break, continue, return)

* break en Kotlin

fun breakStatement() {

while (true) {

val s = readlnOrNull()

if (s.isNullOrEmpty()) break

println(s)

}

}

* continue en Kotlin

fun continueStatement(args: Array<String>) {

for (arg in args) {

if (arg.startsWith("/")) continue

println(arg)

}

}

* return en Kotlin

fun add(a: Int, b: Int): Int {

return a + b

}

fun returnStatement() {

println(add(1, 2))

return

}

### Manejo de Excepciones (try-catch-finally)

lanzar una excepción en Kotlin

fun divide(x: Double, y: Double): Double {

if (y == 0.0) throw ArithmeticException("Cannot divide by zero")

return x / y

}

Capturar Excepciones (try-catch)

fun tryCatch(args: Array<String>) {

try {

if (args.size != 2) {

throw IllegalArgumentException("Two numbers required")

}

val x = args[0].toDouble()

val y = args[1].toDouble()

println(divide(x, y))

} catch (e: IllegalArgumentException) {

println(e.message)

} finally {

println("Goodbye!")

}

}