Programación Móvil

Declaración de Variables

En Kotlin, las variables se pueden declarar usando dos palabras clave: var y val.

var: Se utiliza para declarar variables mutables, es decir, su valor puede cambiar después de ser inicializado.

val: Se utiliza para declarar variables inmutables, es decir, su valor no puede cambiar una vez que se ha asignado.

Cómo Declarar una Variable

La sintaxis básica para declarar una variable en Kotlin es:

var nombreVariable: Tipo = valor

val nombreVariable: Tipo = valor

Declaración de Variables

Tipos de Datos Comunes

Kotlin soporta varios tipos de datos, entre los más comunes se encuentran:

- Números: Int, Long, Float, Double
- Caracteres: Char
- Cadenas: String
- Booleanos: Boolean

```kotlin
// Declaración de una variable mutable
var nombre: String = "Juan"
nombre = "Pedro" // Esto es válido

// Declaración de una variable inmutable
val edad: Int = 25
edad = 30 // Esto no es válido y dará un error
```

```kotlin
val entero: Int = 10
val largo: Long = 100L
val decimal: Double = 10.5
val caracter: Char = 'A'
val cadena: String = "Hola, Kotlin!"
val booleano: Boolean = true
```

Inferencia de Tipos de Variables

Kotlin es un lenguaje de tipo estático, pero tiene la capacidad de inferir el tipo de una variable a partir del valor que se le asigna.

```
val nombre = "Kotlin" // El compilador infiere que el tipo es String
val numero = 42 // El compilador infiere que el tipo es Int
```

```
+ (más)
- (menos)
* (producto)
/ (división)
% (resto de una división)  Ej.:  x = 13 % 5  {se guarda 3}
```

Ejercicio con Variables

- Declara dos variables para almacenar el ancho (5.0) y el alto(10.0) del rectángulo. Usa **var** para que estas variables puedan ser modificadas.
- Declara una variable para almacenar el área del rectángulo. Usa **val** ya que el área no cambiará una vez calculada.
- Calcula el área del rectángulo (ancho * alto) y almacénalo en la variable del área.
- Imprime el ancho, el alto y el área del rectángulo.
- Modifica los valores de ancho y alto después de su declaración inicial y vuelve a calcular el área. Imprime los nuevos valores.
- Declara variables adicionales para calcular y almacenar el perímetro del rectángulo (2 * (ancho + alto)). Imprime el perímetro.

Ejercicio con Variables

proyecto2.kt

```kotlin
fun main() {
    var ancho: Double = 5.0
    var alto: Double = 10.0
    var area: Double = ancho * alto

    println("Ancho del rectángulo: $ancho")
    println("Alto del rectángulo: $alto")
    println("Área del rectángulo: $area")

    ancho = 7.0
    alto = 12.0
    area = ancho * alto

    println("Nuevo ancho del rectángulo: $ancho")
    println("Nuevo alto del rectángulo: $alto")
    println("Nueva área del rectángulo: $area")

    val perimetro: Double = 2 * (ancho + alto)

    println("Perímetro del rectángulo: $perimetro")
}
```

- Para mostrar por la Consola el contenido de la variable $resultado utilizamos la función println y dentro del String que muestra donde queremos que aparezca el contenido de la variable le antecedimos el caracter $
- En Kotlin si no vamos a utilizar el parámetro que llega a la función 'main' podemos obviar el mismo:

Entrada de datos por teclado en la Consola

- readln() es una función en Kotlin que se utiliza para leer una línea de entrada desde la consola, readln() siempre devuelve un String no nulo.

```kotlin
fun main() {
    print("Ingrese primer valor:")
    val valor1 = readln().toInt()
    print("Ingrese segundo valor:")
    val valor2 = readln().toInt()
    val suma = valor1 + valor2
    println("La suma de $valor1 y $valor2 es $suma")
    val producto = valor1 * valor2
    println("El producto de $valor1 y $valor2 es $producto")
}
```

Estructura condicional if

```
if (condición) {
    // Código a ejecutar si la condición es verdadera
} else {
    // Código a ejecutar si la condición es falsa
}
```

```
>  (mayor)
<  (menor)
>= (mayor o igual)
<= (menor o igual)
== (igual)
!= (distinto)
```

Operador `&&` (AND)

Este operador devuelve `true` solo si ambas condiciones son verdaderas.

Operador `||` (OR)

Este operador devuelve `true` si al menos una de las condiciones es verdadera.

Operador `!` (NOT)

Este operador invierte el valor de la condición: `true` se convierte en `false` y viceversa.

Estructura condicional if

```kotlin
fun main() {
    print("Ingrese el sueldo del empleado:")
    val sueldo = readln().toDouble()
    if (sueldo > 3000) {
        println("Debe pagar impuestos")
    }
    else{
        println("No Debe pagar impuestos")

    }
}
```

Estructura condicional if

```kotlin
val a = true
val b = false

if (a && b) {
    println("Ambas condiciones son verdaderas.")
} else {
    println("Al menos una de las condiciones es falsa.")
}
```

```kotlin
val a = true
val b = false

if (a || b) {
    println("Al menos una de las condiciones es verdadera.")
} else {
    println("Ambas condiciones son falsas.")
}
```

```kotlin
val a = true

if (!a) {
    println("La condición es falsa.")
} else {
    println("La condición es verdadera.")
}
```

```kotlin
val a = true
val b = false
val c = true

if ((a && b) || c) {
    println("La condición compuesta es verdadera.")
} else {
    println("La condición compuesta es falsa.")
}
```

Estructura condicional if como expresión

- En Kotlin, la estructura condicional if puede ser usada como expresión, lo que significa que puede devolver un valor. Esto permite asignar el resultado de una condición a una variable directamente.

```
val resultado = if (condición) {
    // Código a ejecutar si la condición es verdadera
    valorSiVerdadero
} else {
    // Código a ejecutar si la condición es falsa
    valorSiFalso
}
```

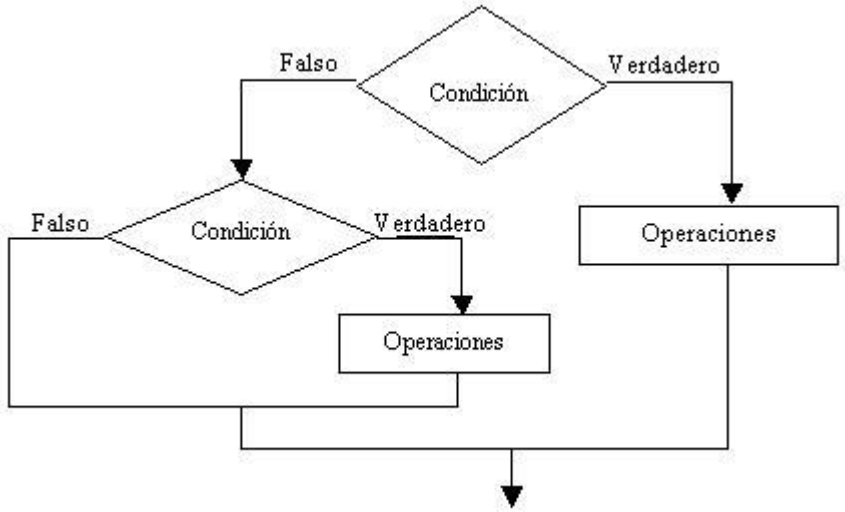
```
val numero = 10
val esPar = if (numero % 2 == 0) "Par" else "Impar"

println(esPar) // Imprime: Par
```

```
fun main(parametro: Array<String>) {
    print("Ingrese primer valor:")
    val valor1 = readln().toInt()
    print("Ingrese segundo valor:")
    val valor2 = readln().toInt()
    val mayor = if (valor1 > valor2) valor1 else valor2
    println("El mayor entre $valor1 y $valor2 es $mayor")
}
```

Estructuras condicionales anidadas

- Decimos que una estructura condicional es anidada cuando por la rama del verdadero o el falso de una estructura condicional hay otra estructura condicional




```kotlin
val numero = 10

if (numero > 0) {
    if (numero % 2 == 0) {
        println("El número es positivo y par.")
    } else {
        println("El número es positivo e impar.")
    }
} else {
    println("El número es negativo o cero.")
}
```

Estructura condicional when

- En Kotlin, when es una estructura que permite evaluar una expresión y ejecutar diferentes bloques de código dependiendo del valor de esa expresión. Es similar a switch en otros lenguajes de programación

```
when (expresión) {
    valor1 -> {
        // Código a ejecutar si la expresión es igual a valor1
    }
    valor2 -> {
        // Código a ejecutar si la expresión es igual a valor2
    }
    else -> {
        // Código a ejecutar si la expresión no coincide con ningún valor anterior
    }
}
```

```
fun main() {
    print("Ingrese un valor entero entre 1 y 5:")
    val valor = readln().toInt()
    when (valor) {
        1 -> print("uno")
        2 -> print("dos")
        3 -> print("tres")
        4 -> print("cuatro")
        5 -> print("cinco")
        else -> print("valor fuera de rango")
    }
}
```

Kotlin

SENA

Estructura condicional when sin argumento

```kotlin
fun main() {
    print("Ingrese coordenada x del punto:")
    val x = readln().toInt()
    print("Ingrese coordenada y del punto:")
    val y = readln().toInt()
    when {
        x > 0 && y > 0 -> println("Primer cuadrate")
        x < 0 && y > 0 -> println("Segundo cuadrante")
        x < 0 && y < 0 -> println("Tercer cuadrante")
        x > 0 && y < 0 -> println("Cuarto cuadrante")
        else -> println("El punto se encuentra en un eje")
    }


}
```

Estructura repetitiva while

- En Kotlin, la estructura repetitiva while se utiliza para ejecutar un bloque de código repetidamente mientras una condición booleana específica sea verdadera.

```
while (condición) {
    // Código a ejecutar
    // mientras la condición sea verdadera
}
```

```
fun main() {
    var contador = 1

    while (contador <= 5) {
        println("Contador: $contador")
        contador++
    }

}
```

Estructura repetitiva do/while

- En Kotlin, la estructura repetitiva do/while se utiliza para ejecutar un bloque de código al menos una vez y luego repetirlo mientras una condición específica sea verdadera..

```
do {
    // Código que se ejecuta al menos una vez
    // y luego mientras la condición sea verdadera
} while (condición)
```

```
fun main() {
    var contador = 1

    do {
        println("Contador: $contador")
        contador++
    } while (contador <= 5)

}
```

Estructura repetitiva for y expresiones de rango

- Las expresiones de rango se utilizan frecuentemente en Kotlin para generar secuencias de valores numéricos o caracteres.

```
for (variable in inicio..fin) {
    // Código que se ejecuta para cada valor
    // dentro del rango desde inicio hasta fin
}
```

```
for (variable in inicio..fin step paso) {
    // Código que se ejecuta para cada valor
    // dentro del rango desde inicio hasta fin con el paso especificado
}
```

```
for (variable in inicio downTo fin step paso) {
    // Itera en orden descendente desde inicio hasta fin,
    // ejecutando el código para cada valor con el paso especificado.
}
```

```
for (i in 1..5) {
    println(i)
}
```

```
for (c in 'a'..'z') {
    println(c)
}
```

Funciones

- En Kotlin, las funciones se definen utilizando la palabra clave fun, seguida del nombre del método, los paréntesis donde declararemos los valores de entrada y unas llaves que limitan la función.

```
fun nombreFuncion(parametro1: Tipo, parametro2: Tipo): TipoRetorno {
    // Cuerpo de la función
    // Puede incluir cualquier cantidad de declaraciones y expresiones
    return valorDeRetorno
}
```

```
fun sumar(a: Int, b: Int): Int {
    return a + b
}
```

Funciones

- En Kotlin, las funciones se definen utilizando la palabra clave fun, seguida del nombre del método, los paréntesis donde declararemos los valores de entrada y unas llaves que limitan la función.

```
fun nombreFuncion(parametro1: Tipo, parametro2: Tipo): TipoRetorno {
    // Cuerpo de la función
    // Puede incluir cualquier cantidad de declaraciones y expresiones
    return valorDeRetorno
}
```

```
fun sumar(a: Int, b: Int): Int {
    return a + b
}
```

Funciones

- En Kotlin, las funciones se definen utilizando la palabra clave fun, seguida del nombre del método, los paréntesis donde declararemos los valores de entrada y unas llaves que limitan la función.

```
fun nombreFuncion(parametro1: Tipo, parametro2: Tipo): TipoRetorno {
    // Cuerpo de la función
    // Puede incluir cualquier cantidad de declaraciones y expresiones
    return valorDeRetorno
}
```

```
1  fun main() {
2      val res= summary( a: 8, b: 7)
3      println(res)
4  }
5
6  fun summary(a: Int, b: Int): Int {
7      return a + b
8  }
9
```

Funciones: con una única expresión

- Las funciones de una única expresión se pueden expresar en Kotlin sin el bloque de llaves y mediante una asignación indicar el valor que retorna.

```kotlin
fun retornarSuperficie(lado: Int) = lado * lado

fun main() {
    print("Ingrese el valor del lado del cuadrado:")
    val la = readln().toInt()
    println("La superficie del cuadrado es ${retornarSuperficie(la)}")
}
```

- Disponemos el operador = y seguidamente la expresión, en este caso el producto del parámetro lado por si mismo.

- No hace falta indicar el tipo de dato que retorna la función ya que el compilador puede inferir que del producto lado * lado se genera un tipo de dato Int..

Funciones: con parámetros con valor por defecto

- En Kotlin, es posible definir parámetros y asignarles un valor predeterminado en la declaración de la función. Al llamar a la función, podemos optar por proporcionar o no un valor para esos parámetros

```kotlin
fun saludar(nombre: String = "Amigo", saludo: String = "Hola") {
    println("$saludo, $nombre!")
}

fun main() {
    saludar()
    saludar("Carlos")
    saludar("Ana", "Buenos días")
}

```

Funciones: llamada a la función con argumentos nombrados

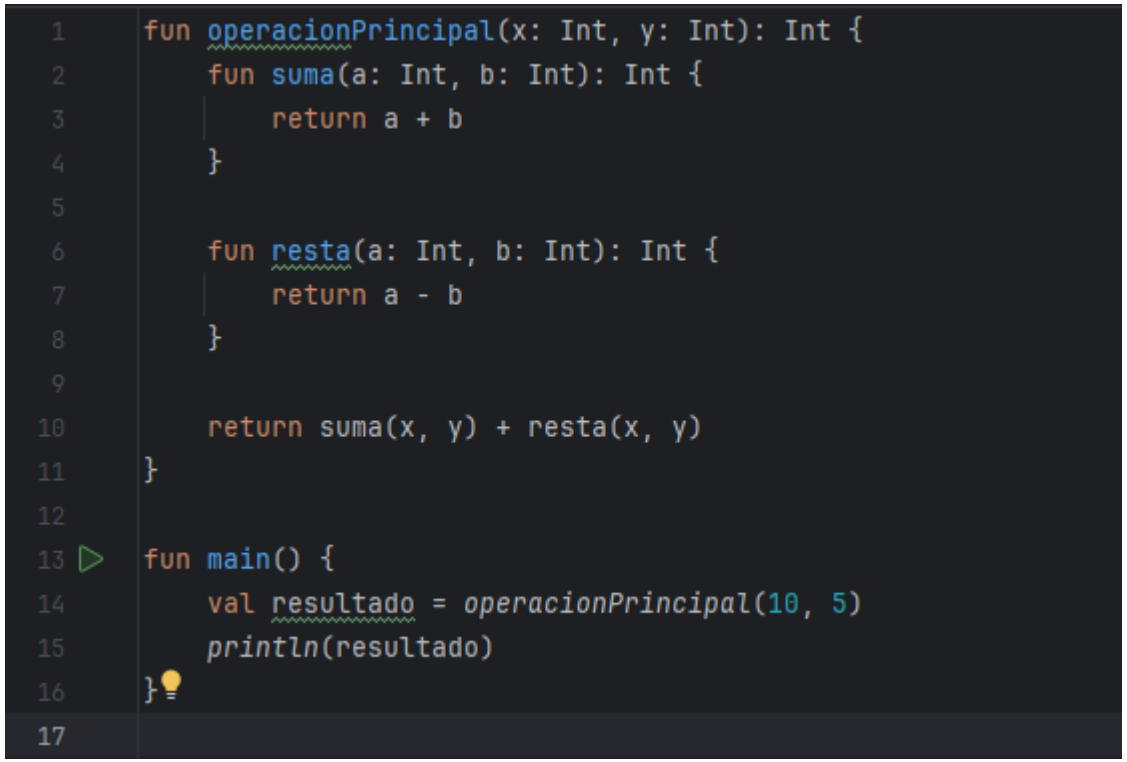
- Esta característica de Kotlin nos permite invocar la función en cualquier orden para los parámetros, siempre que especifiquemos el nombre del parámetro junto con el valor correspondiente en la llamada.

```
fun crearUsuario(nombre: String, edad: Int, ciudad: String) {
    println("Nombre: $nombre, Edad: $edad, Ciudad: $ciudad")
}

fun main() {
    crearUsuario(nombre = "Ana", edad = 30, ciudad = "Madrid")
    crearUsuario(ciudad = "Barcelona", nombre = "Luis", edad = 25)
}
```

Funciones: internas o locales

- Kotlin admite la definición de funciones locales o internas, lo que significa que puedes declarar una función dentro del cuerpo de otra función. Esta capacidad permite organizar el código de manera más estructurada y encapsular la lógica que solo es relevante dentro del contexto de la función externa. Las funciones locales pueden acceder a las variables y parámetros de la función que las contiene, facilitando la reutilización y la claridad del código.



```kotlin
fun operacionPrincipal(x: Int, y: Int): Int {
    fun suma(a: Int, b: Int): Int {
        return a + b
    }

    fun resta(a: Int, b: Int): Int {
        return a - b
    }

    return suma(x, y) + resta(x, y)
}

fun main() {
    val resultado = operacionPrincipal(10, 5)
    println(resultado)
}
```

Arrays(Arreglos) en Kotlin

- En Kotlin, los arreglos son estructuras que permiten almacenar múltiples valores del mismo tipo en una sola variable.

```kotlin
fun main() {
    val numeros = arrayOf(1, 2, 3, 4, 5)

    println("Primer elemento: ${numeros[0]}")
    println("Tamaño del arreglo: ${numeros.size}")

    numeros[0] = 10

    println("Arreglo modificado:")
    for (numero in numeros) {
        println(numero)
    }

    for ((posicion, valor) in numeros.withIndex()) {
        println("La posición $posicion contiene el valor $valor")
    }
}
```

Arrays(Arreglos) en Kotlin

```kotlin
fun main() {
    numeros[1] = 2
    numeros[2] = 3

    // Arreglo de Strings
    val nombres = Array(3) { "" }
    nombres[0] = "Ana"
    nombres[1] = "Luis"
    nombres[2] = "María"

    // Arreglo de Doubles
    val precios = DoubleArray(3)
    precios[0] = 10.5
    precios[1] = 23.99
    precios[2] = 5.75


    // Arreglo de Booleans
    val flags = BooleanArray(2)
    flags[0] = true
    flags[1] = false


    // imprimir sus contenidos utilizando
    // la función joinToString() para convertir
    // los arreglos en cadenas legibles.
    println("Numeros: ${numeros.joinToString()}")
    println("Nombres: ${nombres.joinToString()}")
    println("Precios: ${precios.joinToString()}")
    println("Flags: ${flags.joinToString()}")
}
```

Listas en Kotlin

- Las colecciones pueden dividirse en dos categorías principales: mutables e inmutables. Esto significa que hay colecciones que se pueden modificar (mutables) y colecciones que solo se pueden leer (inmutables).

```kotlin
fun main() {
    val numerosInmutables = listOf(1, 2, 3, 4, 5)
    println("Lista Inmutable: $numerosInmutables")

    val numerosMutables = mutableListOf(1, 2, 3, 4, 5)
    numerosMutables.add(6)
    numerosMutables[0] = 10
    println("Lista Mutable: $numerosMutables")
}
```

Listas en Kotlin Métodos

```kotlin
fun main() {
    // Declaración de una lista mutable
    val edades = mutableListOf(23, 67, 12, 35, 12)

    // Imprimir la lista inicial
    println("Lista inicial: $edades")

    // Agregar un elemento
    edades.add(45)
    println("Después de agregar 45: $edades")

    // Agregar un elemento en una posición especifica
    edades.add(2, 50)
    println("Después de agregar 50 en el índice 2: $edades")

    // Eliminar un elemento por valor
    edades.remove(12)
    println("Después de eliminar 12: $edades")

    // Eliminar un elemento por índice
    edades.removeAt(1)
    println("Después de eliminar el elemento en el índice 1: $edades")
```

Listas en Kotlin
Métodos

```
    // Modificar un elemento por índice
    edades[0] = 30
    println("Después de modificar el primer elemento a 30: $edades")

    // Buscar el índice de un elemento
    val indice = edades.indexOf(50)
    println("Índice del elemento 50: $indice")

    // Verificar si la lista contiene un elemento
    val contiene45 = edades.contains(45)
    println("¿La lista contiene 45?: $contiene45")

    // Obtener el tamaño de la lista
    val tamaño = edades.size
    println("Tamaño de la lista: $tamaño")

    // Ordenar la lista
    edades.sort()
    println("Lista ordenada: $edades")

    // Vaciar la lista
    edades.clear()
    println("Lista después de vaciarla: $edades")
}
```

Clase en Kotlin

```
class [nombre de la clase] {
  [propiedades de la clase]
  [métodos o funciones de la clase]
}
```

```kotlin
fun main(parametro: Array<String>) {
    val persona1: Persona
    persona1 = Persona()
    persona1.inicializar("Juan", 12)
    persona1.imprimir()
    persona1.esMayorEdad()

    val persona2: Persona
    persona2 = Persona()
    persona2.inicializar("Ana", 50)
    persona2.imprimir()
    persona2.esMayorEdad()
}
```

```kotlin
class Persona {
    var nombre: String = ""
    var edad: Int = 0

    fun inicializar(nombre: String, edad: Int) {
        this.nombre = nombre
        this.edad = edad
    }

    fun imprimir() {
        println("Nombre: $nombre y tiene una edad de $edad")
    }

    fun esMayorEdad() {
        if (edad >= 18)
            println("Es mayor de edad $nombre")
        else
            println("No es mayor de edad $nombre")
    }
}
```

Clase en Kotlin Constructor

```kotlin
class Persona constructor(nombre: String, edad: Int) {
    var nombre: String = nombre
    var edad: Int = edad

    fun imprimir() {
        println("Nombre: $nombre y tiene una edad de $edad")
    }

    fun esMayorEdad() {
        if (edad >= 18)
            println("Es mayor de edad $nombre")
        else
            println("No es mayor de edad $nombre")
    }
}

fun main(parametro: Array<String>) {
    val persona1 = Persona("Juan", 12)
    persona1.imprimir()
    persona1.esMayorEdad()
}
```

Clase en Kotlin Modificadores de Acceso

```kotlin
class Operaciones {
    private var valor1: Int = 0
    private var valor2: Int = 0

    fun cargar() {
        print("Ingrese primer valor: ")
        valor1 = readln().toInt()
        print("Ingrese segundo valor: ")
        valor2 = readln().toInt()
        sumar()
        restar()
    }

    private fun sumar() {
        val suma = valor1 + valor2
        println("La suma de $valor1 y $valor2 es $suma")
    }

    private fun restar() {
        val resta = valor1 - valor2
        println("La resta de $valor1 y $valor2 es $resta")
    }
}
```

```kotlin
fun main(parametro: Array<String>) {
    val operaciones1 = Operaciones()
    operaciones1.cargar()
}
```

Clase en Kotlin Métodos get() y set()

```kotlin
class Persona {
    var nombre: String = ""
        get() = field // método get
        set(value) {
            field = value // método set
        }

    var edad: Int = 0
        get() = field // método get
        set(value) {
            field = if (value >= 0) value else 0 // método set con validación
        }
}

fun main() {
    val persona = Persona()
    persona.nombre = "Juan" // Usando el método set
    persona.edad = 25 // Usando el método set
    println("Nombre: ${persona.nombre}") // Usando el método get
    println("Edad: ${persona.edad}") // Usando el método get

    persona.edad = -5 // Intentando asignar un valor negativo
    println("Edad después de asignar val negativo: ${persona.edad}") // Usando el método
}
```

Clase en Kotlin Herencia

```kotlin
open class Persona(val nombre: String, val edad: Int) {
    open fun imprimir() {
        println("Nombre: $nombre")
        println("Edad: $edad")
    }
}

class Empleado(nombre: String, edad: Int, val sueldo: Double): Persona(nombre, edad) {
    override fun imprimir() {
        super.imprimir()
        println("Sueldo: $sueldo")
    }

    fun pagaImpuestos() {
        if (sueldo > 3000)
            println("El empleado $nombre paga impuestos")
        else
            println("El empleado $nombre no paga impuestos")
    }
}
```

```kotlin
fun main(parametro: Array<String>) {
    val persona1 = Persona("Jose", 22)
    println("Datos de la persona")
    persona1.imprimir()

    val empleado1 = Empleado("Ana", 30, 5000.0)
    println("Datos del empleado")
    empleado1.imprimir()
    empleado1.pagaImpuestos()
}
```

- ✓ En Kotlin para que una clase sea heredable debemos anteceder la palabra clave open previo a class,
- ✓ Si queremos que un método se pueda reescribir en una subclase debemos anteceder la palabra clave open
- ✓ La herencia la indicamos después de los dos puntos indicando el nombre de la clase de la cual heredamos y pasando inmediatamente los datos del constructor de dicha clase

G R A C I A S

Línea de atención al ciudadano: 01 8000 910270
Línea de atención al empresario: 01 8000 910682

@SENAComunica

www.sena.edu.co