

ARBOL BINARIO

```
import ArbolBinarioCompleto

fun main() {
    val arbol = ArbolBinarioCompleto()
    var arbolConstruido = false
    var opcion: Int?

    do {
        println("\n== menu arbol binario completo ==")
        println("1. insertar elementos")
        println("2. recorrido preorden")
        println("3. recorrido inorden")
        println("4. recorrido postorden")
        println("5. salir")
        print("elige una opcion: ")

        opcion = readLine()?.toIntOrNull()

        when (opcion) {
            1 -> {
                if (arbolConstruido) {
                    println("ya creaste el arbol")
                } else {
                    var niveles: Int? = null
                    while (niveles == null || niveles <= 0) {
                        print("dame los niveles del arbol: ")
                        niveles = readLine()?.toIntOrNull()
                        if (niveles == null || niveles <= 0) {

```

```
fun main() {
    niveles = readLine()?.toIntOrNull()
    if (niveles == null || niveles <= 0) {
        println("entrada invalida niveles > 0")
    }

    val totalNodos = (1 shl niveles) - 1
    val valores = mutableListOf<Int>()
    println("ingresa $totalNodos valores enter despues de cada valor:")

    while (valores.size < totalNodos) {
        print("valor ${valores.size + 1}: ")
        val valor = readLine()?.toIntOrNull()
        if (valor != null) {
            valores.add(valor)
        } else {
            println("valor invalido")
        }
    }

    arbol.construirDesdeLista(valores)
    arbolConstruido = true
    println("arbol construido exitosamente")
}

2 -> {
```

This screenshot shows the `ArbolBinario.kt` file in an IDE. The `main` function is implemented with a `while` loop that continues until the user enters `5`. Inside the loop, there are five menu options:
1. `1 -> { arbol.construirDesdeLista(valores); arbolConstruido = true; println("arbol construido exitosamente"); }`
2. `2 -> { println("\nrecorrido preorden:"); arbol.recorridoPreOrden(); }`
3. `3 -> { println("\nrecorrido inorden:"); arbol.recorridoInOrden(); }`
4. `4 -> { println("\nrecorrido postorden:"); arbol.recorridoPostOrden(); }`
5. `5 -> println("saliendo del programa...");`
The loop is controlled by `while (opcion != 5)`. The IDE interface includes a sidebar with file explorer and tool windows, and a status bar at the bottom showing file encoding (UTF-8) and line endings (CRLF).

```
3 fun main() {  
47     arbol.construirDesdeLista(valores)  
48     arbolConstruido = true  
49     println("arbol construido exitosamente")  
50 }  
51  
52  
53 2 -> {  
54     println("\nrecorrido preorden:")  
55     arbol.recorridoPreOrden()  
56 }  
57  
58 3 -> {  
59     println("\nrecorrido inorden:")  
60     arbol.recorridoInOrden()  
61 }  
62  
63 4 -> {  
64     println("\nrecorrido postorden:")  
65     arbol.recorridoPostOrden()  
66 }  
67  
68 5 -> println("saliendo del programa...")  
69 else -> println("opcion invalida intenta de nuevo")  
70 }  
71 } while (opcion != 5)  
72 }
```

ARBOL BINARIO COMPLETO

This screenshot shows the `ArbolBinarioCompleto.kt` file. It defines a class `ArbolBinarioCompleto` with a private attribute `raiz: NodoArbol?`. The `construirDesdeLista` function takes a `List<Int>` and builds the tree using a queue. It starts by adding the root node. Then, it iterates through the list, adding left and right children to the nodes currently in the queue. The `recorridoInOrden` function is also partially visible at the bottom. The IDE interface is consistent with the previous screenshot, showing the same sidebar and status bar.

```
1 import java.util.LinkedList  
2 import java.util.Queue  
3  
4 class ArbolBinarioCompleto { 2 Usages  
5     private var raiz: NodoArbol? = null 5 Usages  
6  
7     fun construirDesdeLista(valores: List<Int>) { 1 Usage  
8         if (valores.isEmpty()) return  
9  
10        raiz = NodoArbol( valor = valores[0])  
11        val cola: Queue<NodoArbol> = LinkedList()  
12        cola.add(raiz)  
13  
14        var i = 1  
15        while (i < valores.size) {  
16            val actual = cola.poll()  
17            if (i < valores.size) {  
18                actual.izquierdo = NodoArbol( valor = valores[i++])  
19                cola.add(actual.izquierdo!!)  
20            }  
21            if (i < valores.size) {  
22                actual.derecho = NodoArbol( valor = valores[i++])  
23                cola.add(actual.derecho!!)  
24            }  
25        }  
26    }  
27  
28    fun recorridoInOrden() { 11 Usages
```

The screenshot shows an IDE window with the file `ArbolBinarioCompleto.kt` open. The code defines a `class ArbolBinarioCompleto` with two methods: `recorridoInOrden()` and `recorridoPreOrden()`. The `recorridoInOrden()` method calls `inOrden(nodo = raiz)` and prints the result. The `inOrden` method is a private recursive function that prints the value of the node, then recursively calls itself on the left and right children. The `recorridoPreOrden()` method calls `preOrden(nodo = raiz)` and prints the result. The `preOrden` method is a private recursive function that prints the value of the node, then recursively calls itself on the left and right children.

```
4 class ArbolBinarioCompleto { 2 Usages
27
28 fun recorridoInOrden() { 1 Usage
29     inOrden( nodo = raiz)
30     println()
31 }
32
33 private fun inOrden(nodo: NodoArbol?) { 3 Usages
34     if (nodo != null) {
35         inOrden( nodo = nodo.izquierdo)
36         print("${nodo.valor} ")
37         inOrden( nodo = nodo.derecho)
38     }
39 }
40
41 fun recorridoPreOrden() { 1 Usage
42     preOrden( nodo = raiz)
43     println()
44 }
45
46 private fun preOrden(nodo: NodoArbol?) { 3 Usages
47     if (nodo != null) {
48         print("${nodo.valor} ")
49         preOrden( nodo = nodo.izquierdo)
50         preOrden( nodo = nodo.derecho)
51     }
52 }
53 }
```

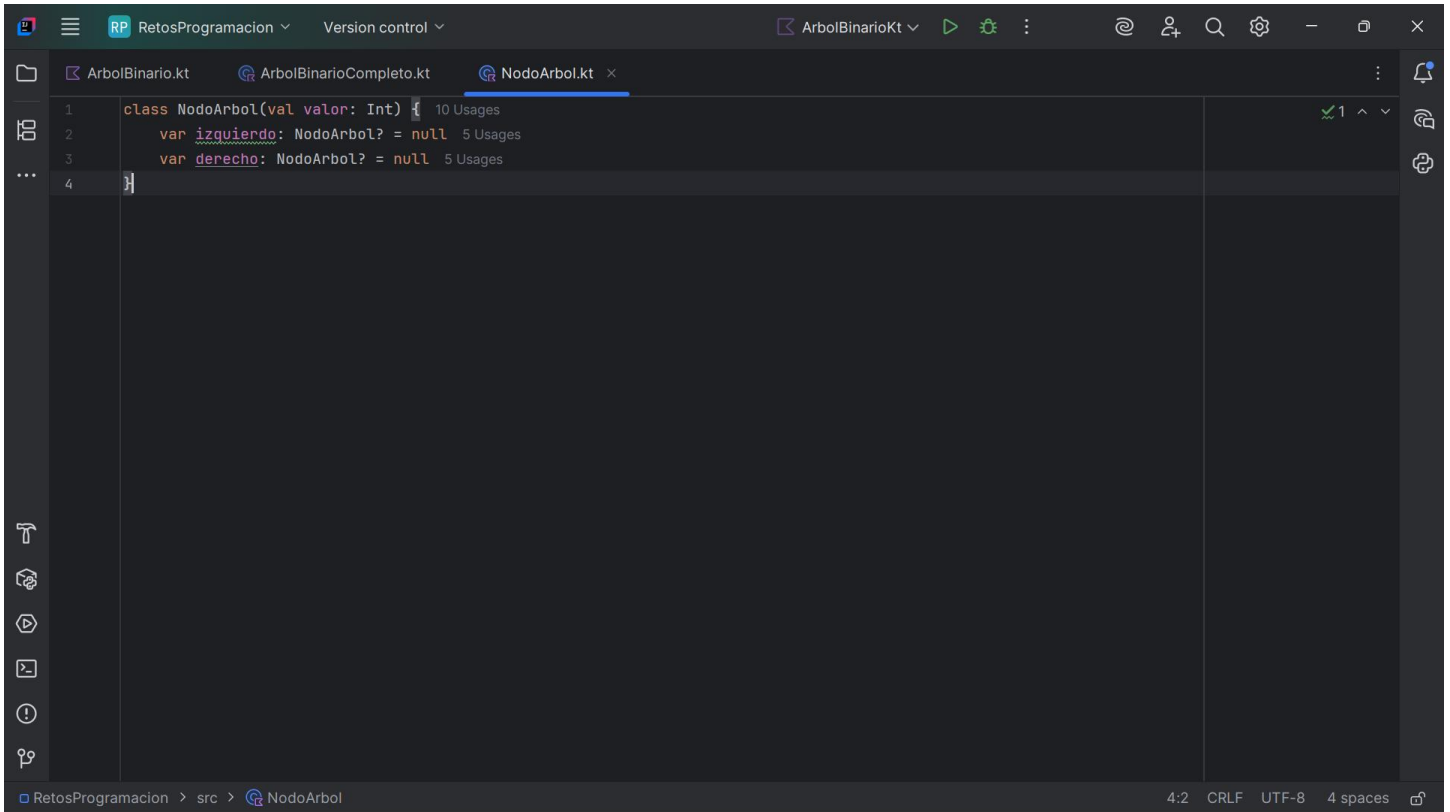
RetosProgramacion > src > ArbolBinarioCompleto 45:1 CRLF UTF-8 4 spaces

The screenshot shows the same IDE window, but the code is scrolled down to show the `recorridoPostOrden()` method. The `recorridoPostOrden()` method calls `postOrden(nodo = raiz)` and prints the result. The `postOrden` method is a private recursive function that recursively calls itself on the left and right children, then prints the value of the node.

```
45
46 private fun preOrden(nodo: NodoArbol?) { 3 Usages
47     if (nodo != null) {
48         print("${nodo.valor} ")
49         preOrden( nodo = nodo.izquierdo)
50         preOrden( nodo = nodo.derecho)
51     }
52 }
53
54 fun recorridoPostOrden() { 1 Usage
55     postOrden( nodo = raiz)
56     println()
57 }
58
59 private fun postOrden(nodo: NodoArbol?) { 3 Usages
60     if (nodo != null) {
61         postOrden( nodo = nodo.izquierdo)
62         postOrden( nodo = nodo.derecho)
63         print("${nodo.valor} ")
64     }
65 }
66 }
```

RetosProgramacion > src > ArbolBinarioCompleto 45:1 CRLF UTF-8 4 spaces

NODO ARBOL

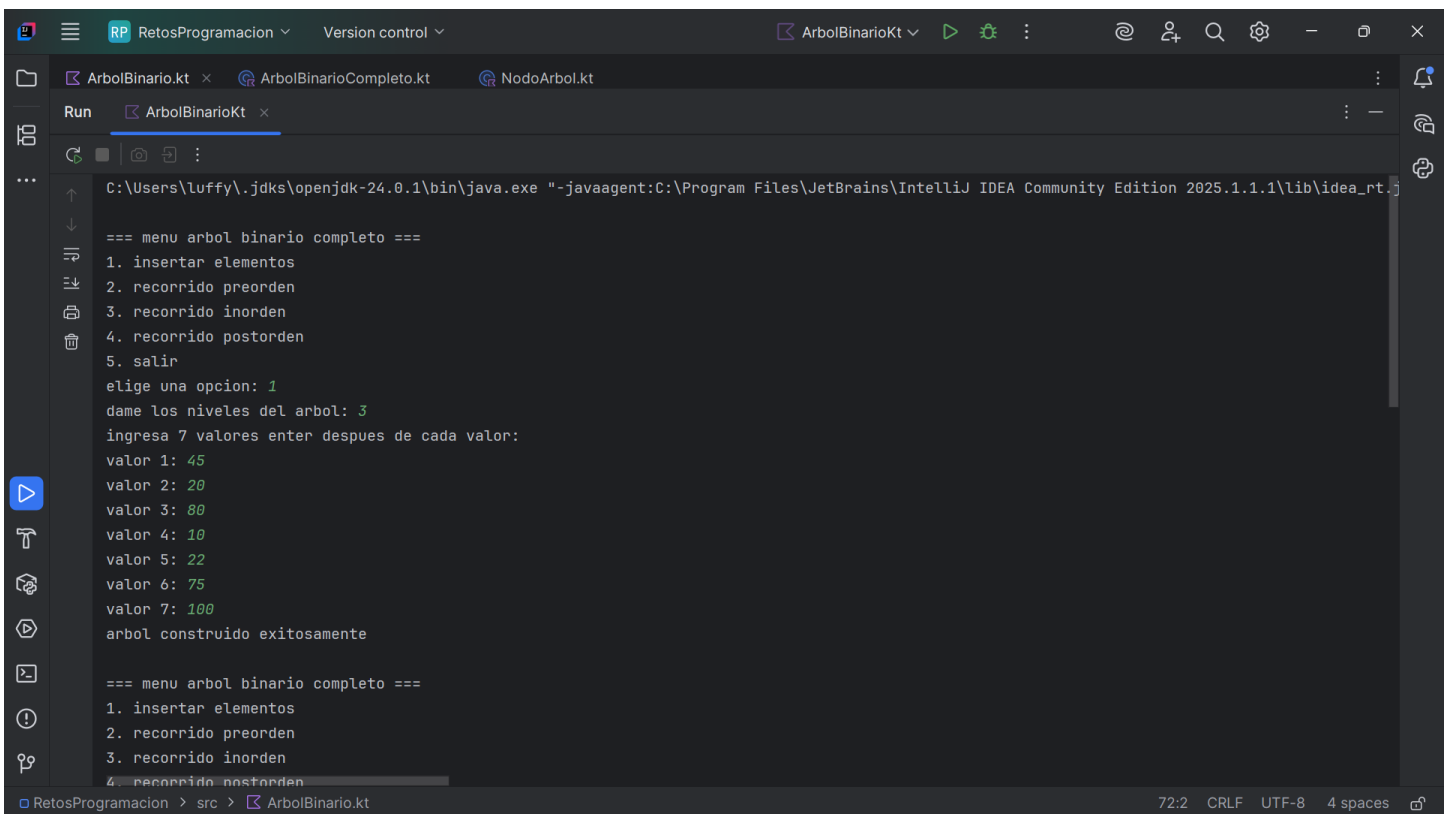


The screenshot shows the IntelliJ IDEA editor with the 'NodoArbol.kt' file open. The code defines a class 'NodoArbol' with a constructor taking an integer 'valor' and two nullable references to 'NodoArbol' for 'izquierdo' and 'derecho'.

```
1 class NodoArbol(val valor: Int) {
2     var izquierdo: NodoArbol? = null
3     var derecho: NodoArbol? = null
4 }
```

The status bar at the bottom indicates the file is in the 'src' directory of the 'RetosProgramacion' project, using the '4:2' encoding, 'CRLF' line endings, 'UTF-8' character set, and '4 spaces' indentation.

CONSOLA



The screenshot shows the IntelliJ IDEA console with the output of a Java program. The program displays a menu for a binary tree, prompts the user to enter values, and prints the resulting tree structure.

```
C:\Users\luffy\.jdk\openjdk-24.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.1.1.1\lib\idea_rt.jar"

=== menu arbol binario completo ===
1. insertar elementos
2. recorrido preorden
3. recorrido inorden
4. recorrido postorden
5. salir
elige una opcion: 1
dame los niveles del arbol: 3
ingresa 7 valores enter despues de cada valor:
valor 1: 45
valor 2: 20
valor 3: 80
valor 4: 10
valor 5: 22
valor 6: 75
valor 7: 100
arbol construido exitosamente

=== menu arbol binario completo ===
1. insertar elementos
2. recorrido preorden
3. recorrido inorden
4. recorrido postorden
```

The status bar at the bottom indicates the file is in the 'src' directory of the 'RetosProgramacion' project, using the '7:2' encoding, 'CRLF' line endings, 'UTF-8' character set, and '4 spaces' indentation.

```
4. recorrido postorden
5. salir
elige una opcion: 2

recorrido preorden:
45 20 10 22 80 75 100

=== menu arbol binario completo ===
1. insertar elementos
2. recorrido preorden
3. recorrido inorden
4. recorrido postorden
5. salir
elige una opcion: 3

recorrido inorden:
10 20 22 45 75 80 100

=== menu arbol binario completo ===
1. insertar elementos
2. recorrido preorden
3. recorrido inorden
4. recorrido postorden
5. salir
elige una opcion: 4
```

```
10 20 22 45 75 80 100

=== menu arbol binario completo ===
1. insertar elementos
2. recorrido preorden
3. recorrido inorden
4. recorrido postorden
5. salir
elige una opcion: 4

recorrido postorden:
10 22 20 75 100 80 45

=== menu arbol binario completo ===
1. insertar elementos
2. recorrido preorden
3. recorrido inorden
4. recorrido postorden
5. salir
elige una opcion: 5
saliendo del programa...

Process finished with exit code 0
```