UNIVERSIDAD POLITÉCNICA DE CHIAPAS

Alumnos: Ayelen Monserrath Rodriguez Flores - 243681 y Fernando Alexis Duran Vazquez - 243827

Grado y grupo: 4° B

Asignatura: Estructura de datos

Docente: ELIAS BELTRAN NATURI

U1ACT03 - LABERINTO GENÉRICO

22.09.2025

Requisitos del programa

- Representación del Laberinto (Arreglos/Tensores):

El requisito es que el laberinto sea una matriz bidimensional (**T[][]**) con elementos de un tipo genérico T.

La forma en que nosotros lo implementamos es que la clase **BuscadorDeLaberinto<T>** está definida con un tipo genérico T. **T[][] laberinto,** que cumple directamente con el requisito. Esto te permite usar el mismo código para laberintos de caracteres, enteros u otros tipos, como se demuestra en el método main con el ejemplo de **Character[][]**

- solución del Problema (Recursividad):

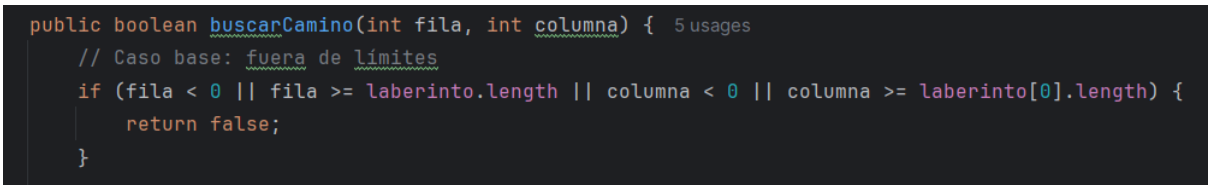
El programa debe implementar la búsqueda del camino usando una función recursiva llamada **buscarCamino** que se mueva en las cuatro direcciones.

La implementación en nuestro programa en el metodo **public boolean buscarCamino(int fila, int columna)** que es la función recursiva. Dentro de este método, se realizan cuatro llamadas a sí mismo **buscarCamino(fila - 1, columna), buscarCamino(fila + 1, columna), buscarCamino(fila, columna - 1), buscarCamino(fila, columna + 1)** para explorar todas las direcciones posibles.

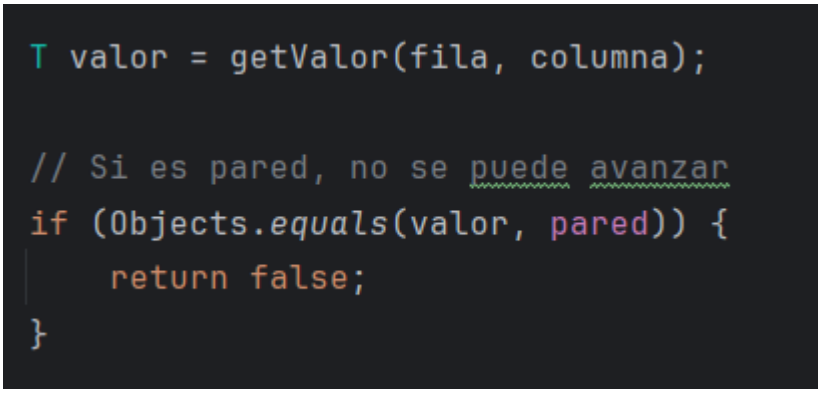
- Casos Base de la Recursión

Caso Base 1 (Fuera de límites/Pared): Si la posición está fuera de los límites o es una pared, la función debe terminar y regresar un valor que impida seguir.

En nuestro caso las primeras líneas del método manejan lo siguiente:



```
public boolean buscarCamino(int fila, int columna) {  5 usages
    // Caso base: fuera de límites
    if (fila < 0 || fila >= laberinto.length || columna < 0 || columna >= laberinto[0].length) {
        return false;
    }
```



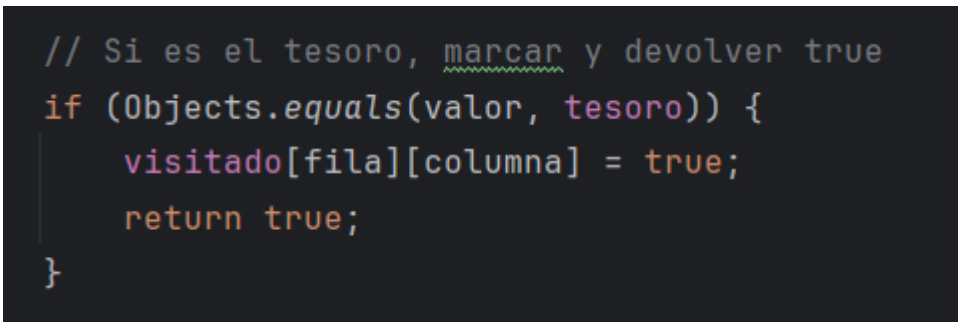
```
T valor = getValor(fila, columna);

// Si es pared, no se puede avanzar
if (Objects.equals(valor, pared)) {
    return false;
}
```

Aquí, si la posición está fuera de rango o es una pared, el método retorna **false**, lo que detiene esa rama de la búsqueda.

Caso Base 2 (Tesoro): Si la posición actual es el tesoro, la función debe terminar y regresar **true**.

Inmediatamente después de verificar si es una pared, el código revisa si ha encontrado el tesoro:



```
// Si es el tesoro, marcar y devolver true
if (Objects.equals(valor, tesoro)) {
    visitado[fila][columna] = true;
    return true;
}
```

Si se encuentra el tesoro, se marca la posición como visitada y se retorna **true**, lo que hace que todas las llamadas recursivas previas en el camino también regresen **true**.

Paso Recursivo y Marcas: Si la posición es un camino válido, el programa debe marcarla como visitada y llamar a la función para las direcciones posibles.

Nuestro código lo implementa de la siguiente manera:

```
// Marcar como visitado (posible parte del camino)
visitado[fila][columna] = true;

// Probar las 4 direcciones: arriba, abajo, izquierda, derecha
boolean encontrado =
        buscarCamino( fila: fila - 1, columna) ||
                buscarCamino( fila: fila + 1, columna) ||
                buscarCamino(fila,  columna: columna - 1) ||
                buscarCamino(fila,  columna: columna + 1);

if (encontrado) {
    return true; // si alguna ruta encontró el tesoro, conservar la marca
}

// Backtracking: si no encontró por aquí, desmarcar y retroceder
visitado[fila][columna] = false;
return false;
```

Aquí, la posición se marca como visitada. Si una de las llamadas recursivas encuentra el tesoro (**encontrado** es **true)**, el método retorna **true**. El uso de **||** (OR) asegura que la búsqueda continúe en otras direcciones solo si la anterior no tuvo éxito. La línea **visitado[fila][columna] = false;** es un paso de *backtracking* que desmarca las celdas si no forman parte de un camino válido.

- Programación Flexible (Datos Genéricos)

La clase debe ser genérica y manejar valores sin saber si son caracteres o números, usando comparaciones como **.equals()**.

Nuestro código logra esto de dos formas:

1. La clase **BuscadorDeLaberinto<T>** es genérica, como se mencionó anteriormente.
2. En lugar de usar **==** o comparaciones específicas para tipos primitivos, utilizas **Objects.equals(valor, pared)** y

Objects.equals(valor, tesoro). Esto es crucial para trabajar con tipos de objetos genéricos, ya que **.equals()** permite una comparación de valor correcta sin importar si **T** es un **Character**, un **Integer** u otro objeto.