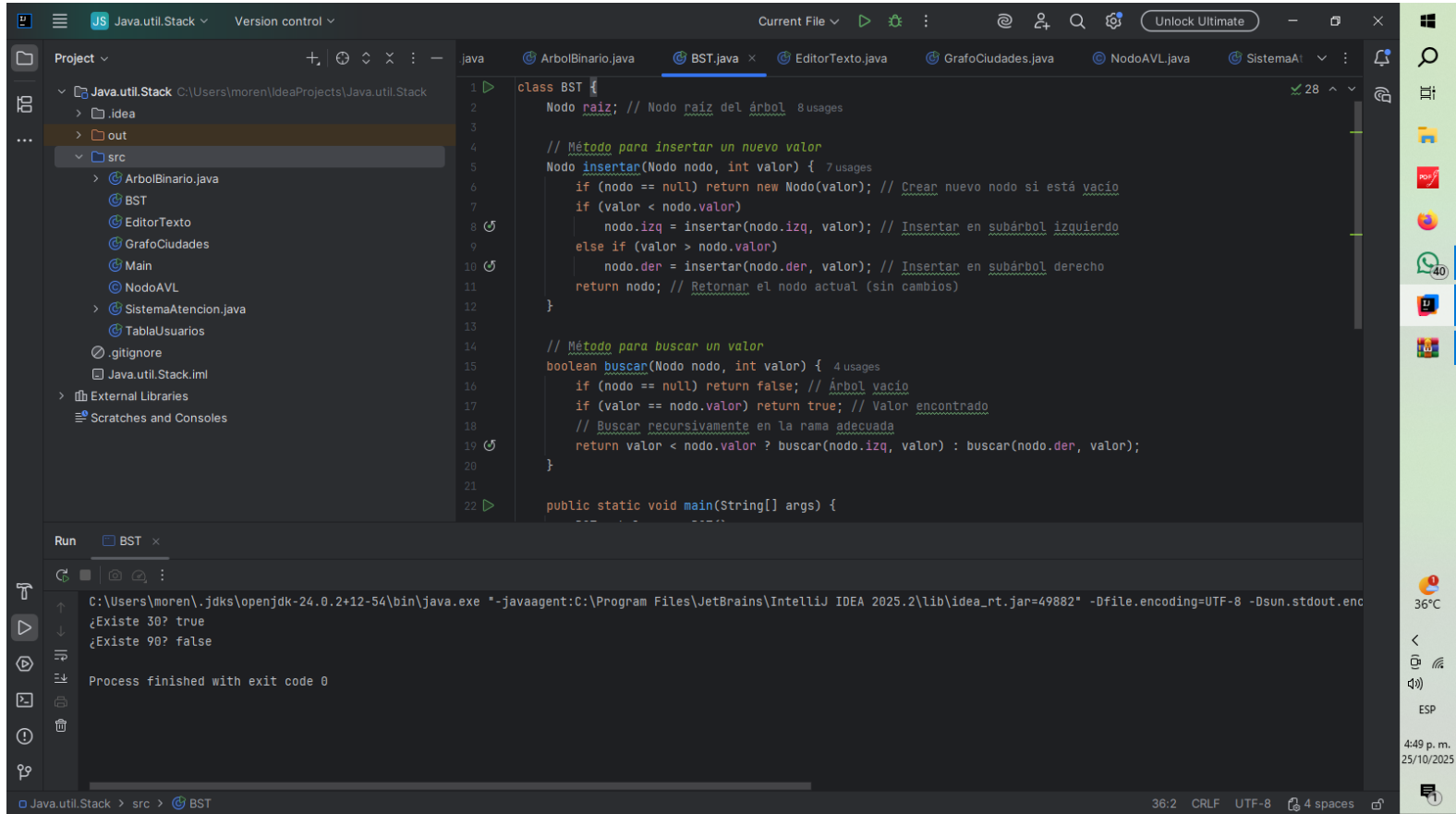


este código en Java lo que hace es crear y recorrer un árbol binario.

Un árbol binario, por si acaso, es una estructura donde cada elemento —al que llamamos nodo— puede tener dos hijos como máximo: uno a la izquierda y otro a la derecha.

Primero se define la clase `Nodo`, que es como el molde para cada parte del árbol. Cada nodo guarda un número (valor) y tiene dos “puertas” para conectarse con otros nodos: una hacia el hijo izquierdo (izq) y otra hacia el hijo derecho (der). Cuando creas un nuevo nodo, por ejemplo `new Nodo(5)`, estás creando una bolita con el número 5 dentro, pero todavía no tiene hijos conectados.

Después viene la clase `ArbolBinario`, que tiene un atributo llamado `raiz`, que es el punto de partida del árbol, el nodo principal. Desde esa raíz se van conectando todos los demás nodos, formando una especie de estructura con ramas.



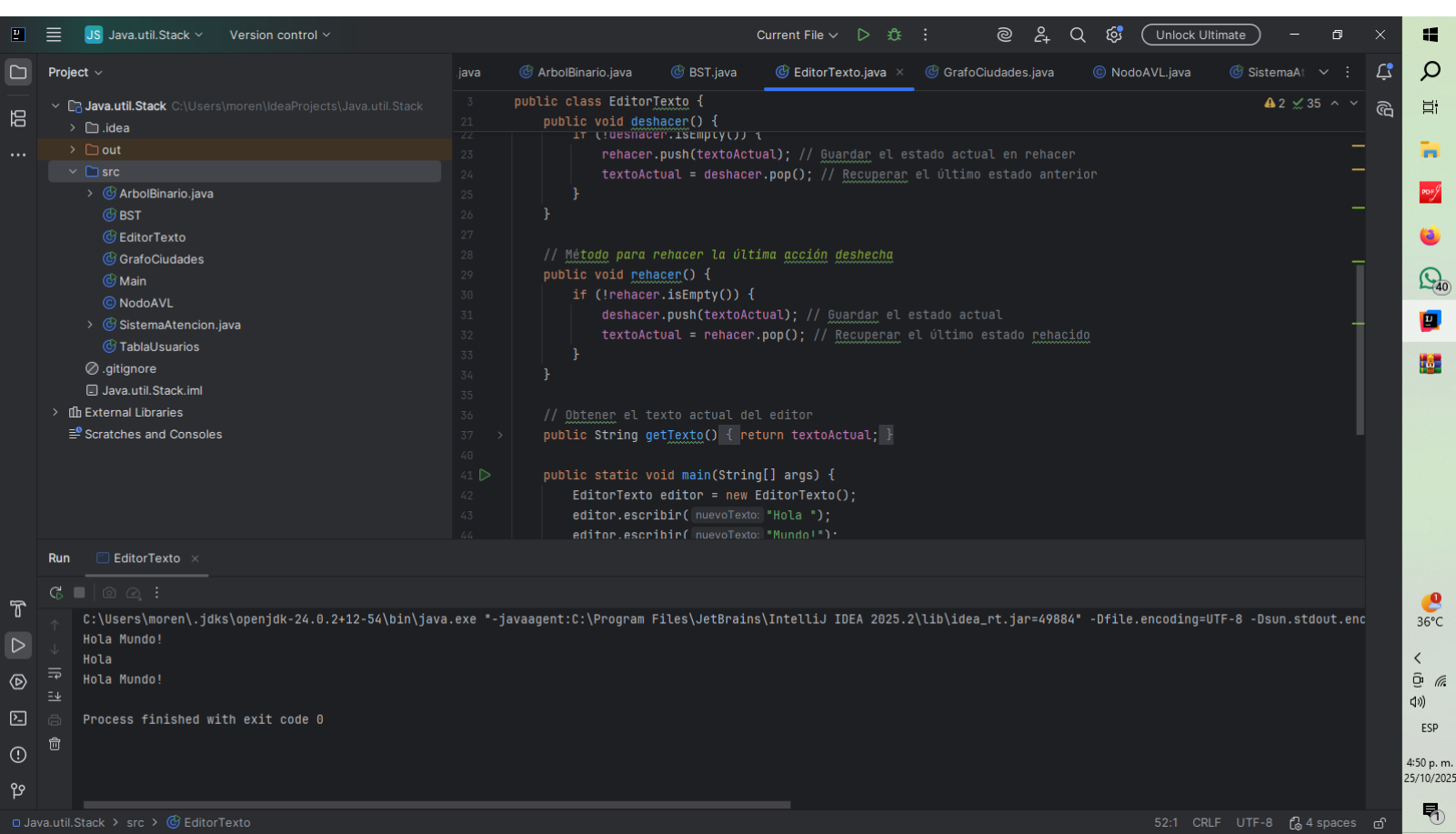
Este código crea un árbol binario de búsqueda (BST), una estructura que guarda números ordenados.

El árbol tiene una raíz, que es el punto principal.

El método insertar coloca los números: si el valor es menor, va a la izquierda; si es mayor, va a la derecha, manteniendo el orden.

El método buscar revisa si un número está en el árbol comparando y bajando por las ramas hasta encontrarlo o no.

En el main, se insertan varios números (50, 30, 70, 20, 40) y luego se prueba si ciertos valores existen, mostrando true o false.



este código es como un editor de texto muy sencillo que deja escribir deshacer y rehacer lo que haces

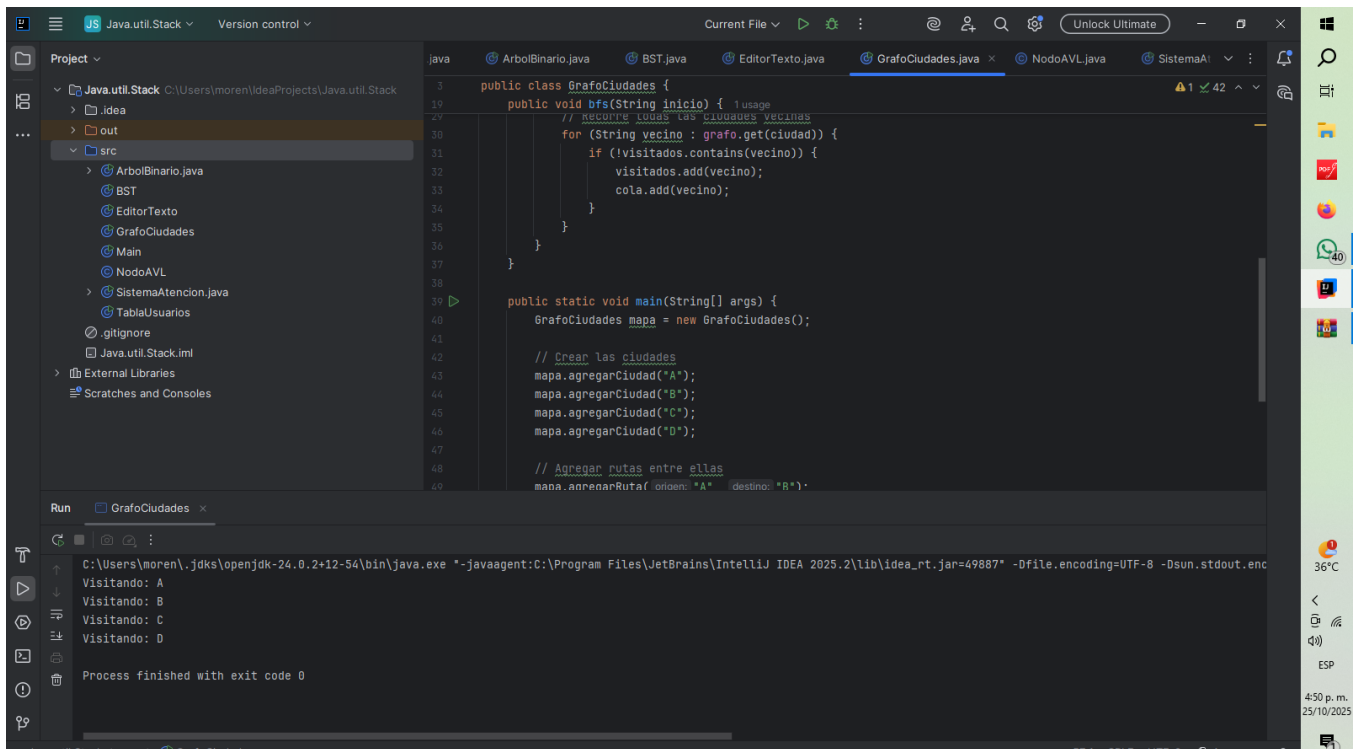
usa dos pilas una llamada deshacer y otra llamada rehacer que son como montones donde se guarda el texto anterior o el texto que se puede recuperar hay una variable llamada textoActual que es donde se guarda lo que estas escribiendo en ese momento

cuando escribes algo nuevo el programa guarda el texto anterior en la pila de deshacer después agrega el nuevo texto al final y borra lo que había en rehacer porque ya cambiaste algo

cuando haces deshacer el programa mira si hay algo guardado y si lo hay pasa el texto actual a rehacer y recupera el texto anterior de deshacer

cuando haces rehacer hace lo contrario o sea saca el texto de rehacer y lo vuelve a poner como el actual

en la parte principal el programa escribe hola luego mundo y lo muestra después hace un deshacer y borra mundo quedando solo hola y luego hace rehacer y vuelve a mostrar hola mundo



este código es como un mapa de ciudades donde cada ciudad puede estar conectada con otras por medio de rutas

entonces lo que hace el programa es guardar todas esas conexiones en algo que se llama grafo que básicamente es una estructura donde cada ciudad tiene una lista de las ciudades a las que se puede llegar

primero hay un metodo para agregar ciudades y otro para conectar una con otra osea
agregar una ruta entre dos ciudades

las rutas son de ida y vuelta asi que si conectas la ciudad a con la b automaticamente tambien se conecta b con a

despues hay un metodo llamado bfs que sirve para recorrer las ciudades empezando desde una y viendo todas las que estan conectadas por niveles osea primero las mas cercanas luego las que estan mas lejos

para eso usa una cola que va guardando las ciudades por visitar y un conjunto para no repetir las que ya se visitaron

en el main o sea la parte principal se crean cuatro ciudades a b c y d despues se

hacen algunas conexiones por ejemplo a con b a con c y b con d

luego el programa empieza el recorrido desde la ciudad a y va visitando las demas segun el orden que encuentra

```
21 public class SistemaAtencion {
22     public static void main(String[] args) {
23
24         PriorityQueue<Cliente> cola = new PriorityQueue<>();
25
26         // Agregar clientes con distintas prioridades
27         cola.add(new Cliente( nombre: "Ana", prioridad: 2));
28         cola.add(new Cliente( nombre: "Carlos", prioridad: 1));
29         cola.add(new Cliente( nombre: "Beto", prioridad: 3));
30
31         // Atender clientes en orden de prioridad
32         while (!cola.isEmpty()) {
33             Cliente c = cola.poll(); // poll() obtiene y elimina el primero
34             System.out.println("Atendiendo a: " + c.nombre);
35         }
36     }
37 }
38
```

Run SistemaAtencion x

C:\Users\moren\.jdk\openjdk-24.0.2+12-54\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2025.2\lib\idea_rt.jar=49897" -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8

Atendiendo a: Carlos
Atendiendo a: Ana
Atendiendo a: Beto

Process finished with exit code 0

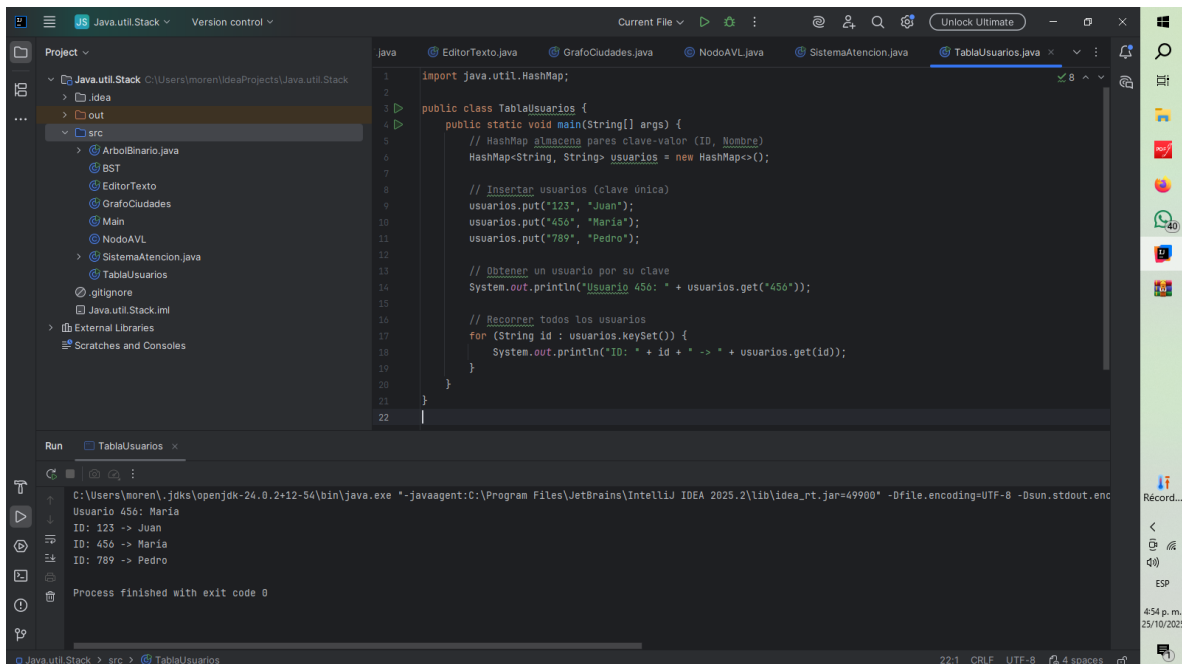
este código es como un sistema para atender personas pero según la prioridad que tengan o sea que no se atienden en el orden que llegan sino dependiendo de quien tiene más importancia

primero hay una clase llamada cliente que tiene dos cosas el nombre y la prioridad mientras más pequeño es el número más prioridad tiene por ejemplo uno es más urgente que dos o tres

luego en el main se usa algo que se llama priorityqueue que es una cola especial que organiza sola los elementos según su prioridad o sea que siempre el primero en salir será el que tenga el número más pequeño

entonces se agregan tres clientes ana con prioridad dos carlos con prioridad uno y beto con prioridad tres

cuando el programa empieza a atender los clientes la cola los organiza y los saca en el orden correcto primero carlos porque tiene prioridad uno luego ana con prioridad dos y por último beto con prioridad tres



The screenshot shows an IDE window with a project named 'Java.util.Stack'. The 'src' folder contains several files, including 'TablaUsuarios.java'. The code in 'TablaUsuarios.java' is as follows:

```
1 import java.util.HashMap;
2
3 public class TablaUsuarios {
4     public static void main(String[] args) {
5         // HashMap almacena pares clave-valor (ID, Nombre)
6         HashMap<String, String> usuarios = new HashMap<>();
7
8         // Insertar usuarios (clave única)
9         usuarios.put("123", "Juan");
10        usuarios.put("456", "Maria");
11        usuarios.put("789", "Pedro");
12
13        // Obtener un usuario por su clave
14        System.out.println("Usuario 456: " + usuarios.get("456"));
15
16        // Recorrer todos los usuarios
17        for (String id : usuarios.keySet()) {
18            System.out.println("ID: " + id + " -> " + usuarios.get(id));
19        }
20    }
21 }
22
```

The 'Run' console at the bottom shows the output of the program:

```
C:\Users\moren\.jdk\openjdk-24.0.2+12-54\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2025.2\lib\idea_rt.jar=49900" -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8
Usuario 456: Maria
ID: 123 -> Juan
ID: 456 -> Maria
ID: 789 -> Pedro
Process finished with exit code 0
```

este código es como una pequeña tabla donde se guardan usuarios con un número que los identifica como si fuera una cédula o un ID. Para eso se usa algo que se llama `HashMap` que básicamente guarda cosas por pares, o sea, una clave y un valor. En este caso, la clave es el ID y el valor es el nombre de la persona.

Entonces, el programa guarda tres usuarios: uno con ID ciento veintitres que es Juan, otro con ID cuatrocientos cincuenta y seis que es María, y otro con ID setecientos ochenta y nueve que es Pedro.

Después, el programa muestra el usuario con el ID cuatrocientos cincuenta y seis, y luego recorre todos los usuarios mostrando el ID y el nombre de cada uno.

```
1 // Nodo para árbol AVL
2 class NodoAVL {
3     int valor, altura;
4     NodoAVL iza, der;
5
6     NodoAVL(int v) {
7         valor = v;
8         altura = 1; // Altura inicial del nodo
9     }
10
11 // Aquí se implementan las rotaciones LL, RR, LR, RL para mantener el balance.
12 // Cada inserción verifica el "factor de balance" = altura izquierda - derecha.
13 // Si el factor > 1 o < -1, se realiza una rotación para equilibrar.
14
15 }
```

este código es como la base de un árbol AVL que es un tipo de árbol que se mantiene siempre balanceado o sea que no se inclina demasiado hacia un lado cada parte del árbol se llama nodo y en este caso cada nodo guarda un número que se llama valor y también tiene dos ramas una a la izquierda y otra a la derecha además tiene algo que se llama altura que sirve para saber que tan alto está ese nodo dentro del árbol

cuando se agregan valores al árbol AVL este revisa algo que se llama factor de balance que básicamente es la diferencia entre la altura del lado izquierdo y el derecho si ese número es mayor que uno o menor que menos uno significa que el árbol está desbalanceado entonces se hacen unos movimientos llamados rotaciones para acomodarlo y dejarlo equilibrado otra vez