INFORME DE LABORATORIO

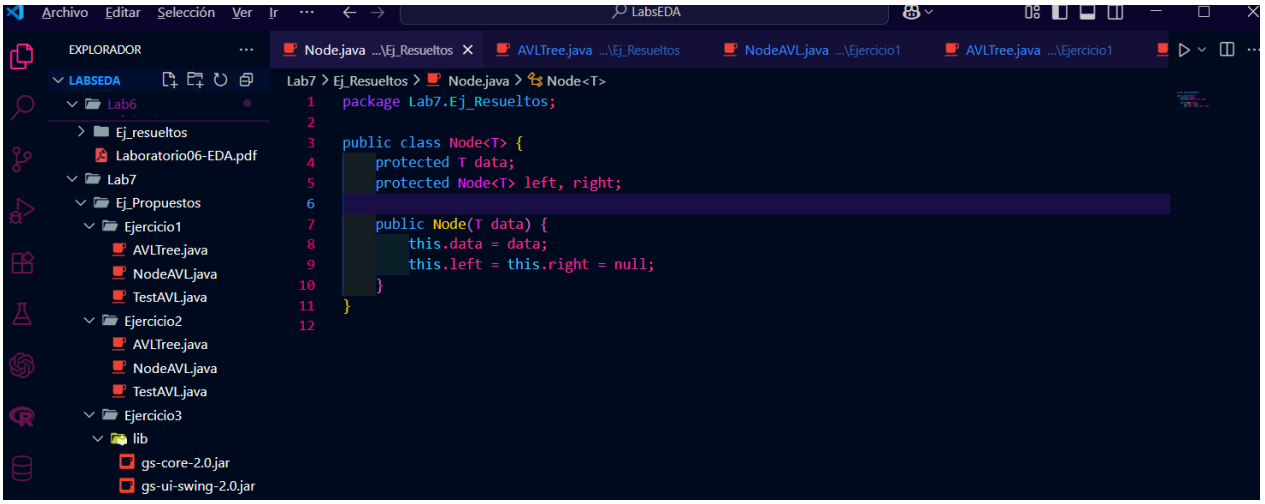
INFORMACIÓN BÁSICA					
ASIGNATURA:	**ESTRUCTURA DE DATOS Y ALGORITMOS**				
TÍTULO DE LA PRÁCTICA:	**ÁRBOL BINARIO AVL BALANCEADO**				
NÚMERO DE PRÁCTICA:	**07**	**AÑO LECTIVO:**	**2025 – A**	**NRO. SEMESTRE:**	**Tercero III**
FECHA DE PRESENTACIÓN	**01/07/2025**	**HORA DE PRESENTACIÓN**	**23:59**		
INTEGRANTE (s): Davila Flores Mathias Dario				**NOTA:**	
DOCENTE(s): ● **Mg. Ing. Rene Alonso Nieto Valencia.** ● **ENLACE GITHUB:** https://github.com/mathiasddf/LabsEDA					

SOLUCIÓN Y RESULTADOS

I. SOLUCIÓN DE EJERCICIOS/PROBLEMAS

a. Ejercicios Resueltos:

i. **Ejercicio 1:** Implementación Nodo Genérico



ii. **Ejercicio 2:** Implementación Clase AVLTree Genérico.

Node.java ...\Ej_Resueltos | AVLTree.java ...\Ej_Resueltos | NodeAVL.java ...\Ejercicio1 | AVLTree.java ...\Ejercicio1

Lab7 > Ej_Resueltos > AVLTree.java > AVLTree<E extends Comparable<E>>

```java
package Lab7.Ej_Resueltos;

public class AVLTree<E extends Comparable<E>> {
    /** Nodo interno con factor de equilibrio. */
    protected class NodeAVL extends Node<E> {
        protected int bf; // balance factor

        public NodeAVL(E data) {
            super(data);
            this.bf = 0;
        }

        @Override
        public String toString() {
            return data.toString();
        }
    }

    private NodeAVL root;
    private boolean height; // flag para cambios de altura

```

iii. **Ejercicio 3 :** Implementación método insertar Genérico

```
22      /** Inserta un elemento, lanzando ItemDuplicated si ya existe. */
23      public void insert(E x) throws ItemDuplicated {
24          this.height = false;
25          this.root = insert(x, this.root);
26      }
27
28      private NodeAVL insert(E x, NodeAVL node) throws ItemDuplicated {
29          if (node == null) {
30              this.height = true;
31              return new NodeAVL(x);
32          }
33
34          int cmp = x.compareTo(node.data);
35          if (cmp == 0) {
36              throw new ItemDuplicated(msg:"El elemento ya existe en el árbol.");
37          } else if (cmp < 0) {
38              node.left = insert(x, (NodeAVL)node.left);
39              if (height) rebalanceAfterInsertLeft(node);
40          } else {
41              node.right = insert(x, (NodeAVL)node.right);
42              if (height) rebalanceAfterInsertRight(node);
43          }
44          return node;
45      }
46
47      private void rebalanceAfterInsertLeft(NodeAVL node) {
48          switch (--node.bf) {
49              case  0: height = false;                break;
50              case -1: height = true;                 break;
51              case -2: node = balanceToRight(node); height = false; break;
52          }
53      }
54
55      private void rebalanceAfterInsertRight(NodeAVL node) {
56          switch (++node.bf) {
```

iv. **Ejercicio 4 :** Implementación método balancear a la izquierda Genérico

```
63      private NodeAVL balanceToLeft(NodeAVL z) {
64          NodeAVL y = (NodeAVL)z.right;
65          if (y.bf >= 0) {
66              // rotación simple izq
67              z.bf = y.bf = 0;
68              return rotateSL(z);
69          }
70          // rotación doble der-izq
71          NodeAVL x = (NodeAVL)y.left;
72          switch (x.bf) {
73              case -1: z.bf = 0; y.bf = 1;  break;
74              case  0: z.bf = y.bf = 0;     break;
75              case  1: z.bf = -1; y.bf = 0; break;
76          }
77          x.bf = 0;
78          z.right = rotateSR(y);
79          return rotateSL(z);
80      }
81
82      private NodeAVL balanceToRight(NodeAVL z) {
83          NodeAVL y = (NodeAVL)z.left;
84          if (y.bf <= 0) {
85              // rotación simple der
86              z.bf = y.bf = 0;
87              return rotateSR(z);
88          }
89          // rotación doble izq-der
90          NodeAVL x = (NodeAVL)y.right;
91          switch (x.bf) {
92              case  1: z.bf = 0; y.bf = -1; break;
93              case  0: z.bf = y.bf = 0;     break;
94              case -1: z.bf = 1; y.bf = 0;  break;
95          }
96          x.bf = 0;
```

v. **Ejercicio 5:** Implementación del método rotación simple a la izquierda Genérico.

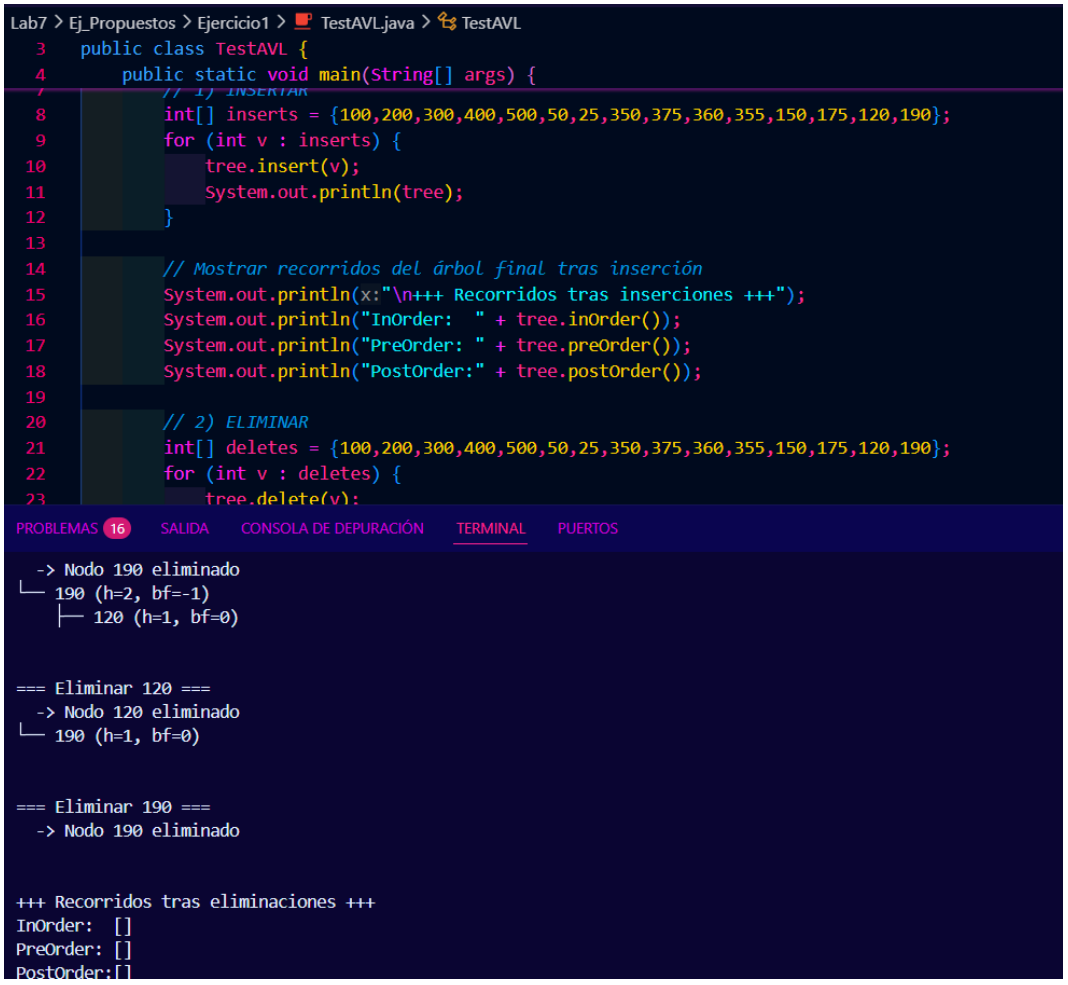
```
101     private NodeAVL rotateSL(NodeAVL node) {
102         NodeAVL p = (NodeAVL)node.right;
103         node.right = p.left;
104         p.left = node;
105         return p;
106     }
107
108     private NodeAVL rotateSR(NodeAVL node) {
109         NodeAVL p = (NodeAVL)node.left;
110         node.left = p.right;
111         p.right = node;
112         return p;
113     }
114
```

b. Ejercicios Propuestos:

i. **EJERCICIO 1:** Mediante la dinámica de un árbol AVL realizar lo siguiente:

- Inserción de los siguientes nodos: 100 – 200 – 300 – 400 – 500 - 50 – 25 – 350 – 375 – 360 – 355 – 150 – 175 – 120 – 190.
- Mostrar los recorridos en inOrder, preOrder y postOrder, del árbol resultante de la inserción.
- Mostrar el árbol resultante de eliminar los nodos: 100 – 200 – 300 – 400 – 500 – 50 – 25 – 350 – 375 – 360 – 355 – 150 – 175 – 120 – 190.
- Mostrar el paso a paso de la inserción y eliminación de cada nodo.
- Indicar las rotaciones utilizadas para balancear el árbol resultante.

En esta implementación se utiliza la clase genérica NodeAVL<T> para representar cada nodo con su clave de tipo T, referencias a hijos izquierdo y derecho y un atributo height que almacena la altura del subárbol; la clase AVLTree<T> gestiona recursivamente la inserción y eliminación, recalculando alturas y evaluando el factor de equilibrio tras cada operación, de modo que cuando el balance supera ±1 aplica automáticamente rotaciones simples o dobles para restaurar la propiedad AVL; finalmente, la clase TestAVL ejecuta la secuencia completa de inserciones y eliminaciones, imprimiendo en consola el estado del árbol (incluyendo altura y factor de equilibrio) después de cada paso y mostrando los recorridos InOrder, PreOrder y PostOrder para verificar la correcta estructura del árbol.

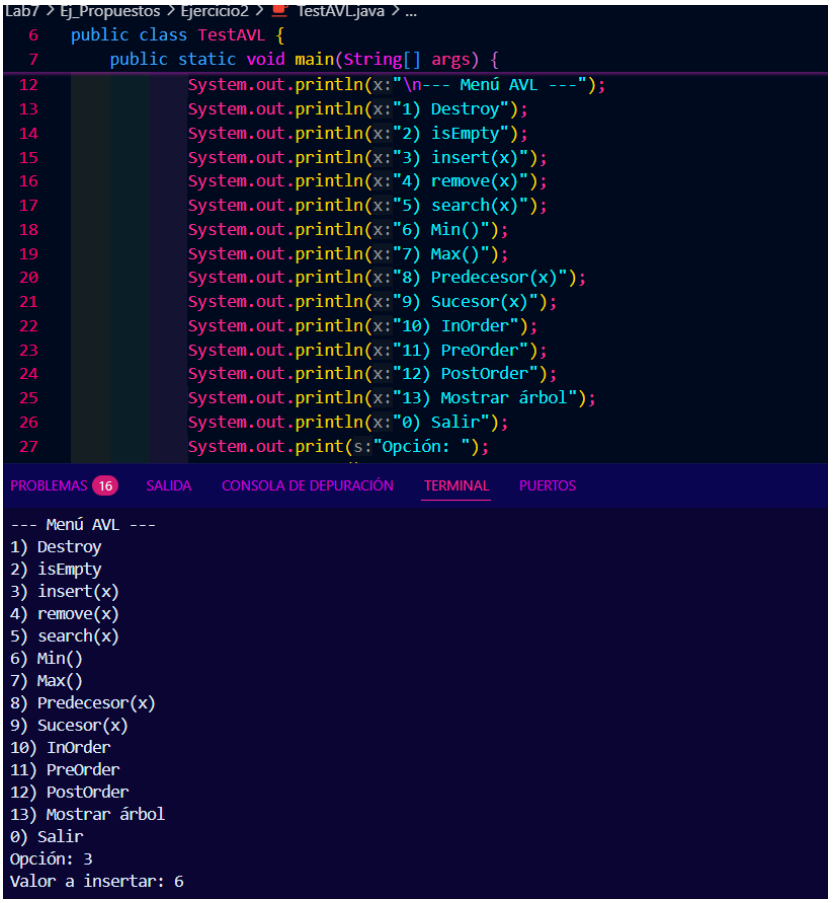


ii. EJERCICIO 2: Implementar un árbol AVL:

- Implementar todas las operaciones: destroy(), isEmpty(), insert(x), remove(x), search(x), Min(), Max(), Predecesor(), Sucesor(), Recorridos: InOrder(), PostOrder(), PreOrder(), balancearIzquierda(), balancearDerecha(), rotacionSimpleIzquierda(), rotacionSimpleDerecha().
- Implementar una clase Test para probar los métodos y mostrar los resultados. utilizando clases y métodos genéricos, utilizando un menú de opciones para todas las operaciones del árbol AVL.

La clase genérica NodeAVL<T> almacena cada clave de tipo T, punteros a hijos y un campo height para el cálculo del factor de equilibrio. AVLTree<T> implementa recursivamente todas las operaciones básicas: inserción y eliminación ajustan altura y calculan bf = altura(derecha) – altura(izquierda) tras cada llamada, y cuando |bf| >1 invocan las rotaciones simples o dobles

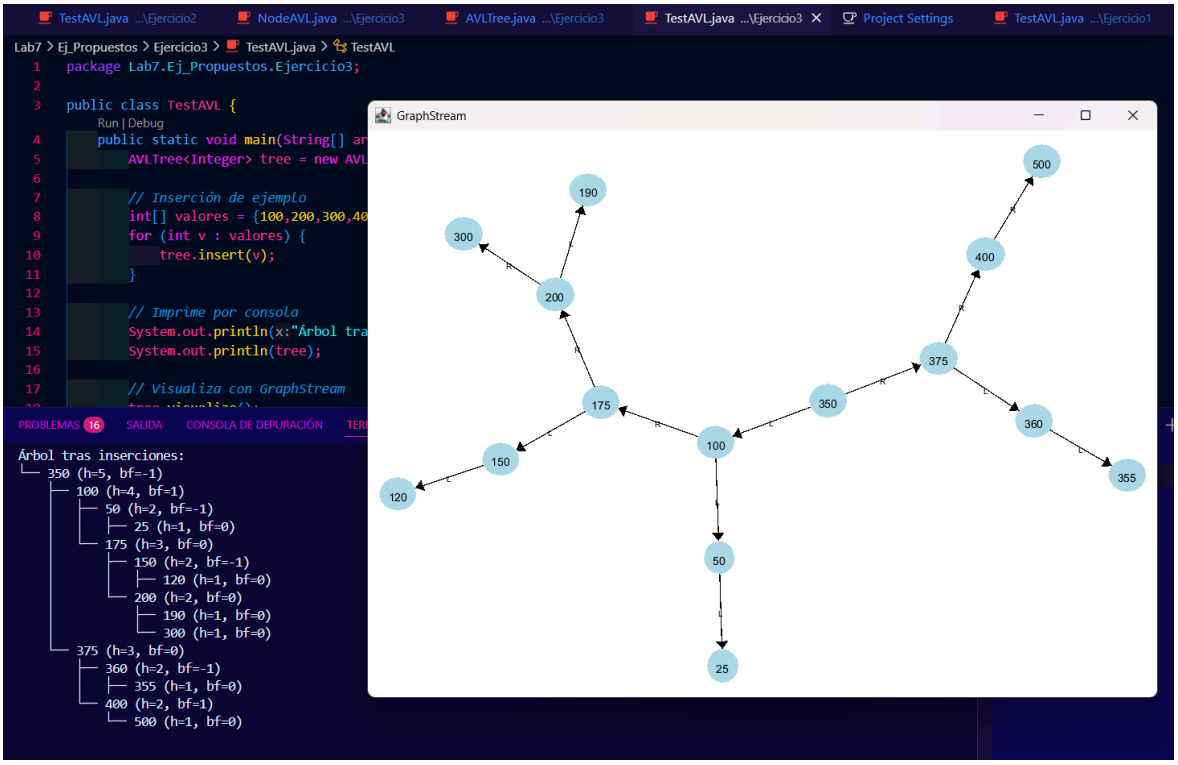
correspondientes para restaurar el balance. Además expone métodos de búsqueda (search), acceso a mínimo y máximo (Min, Max), así como predecesor y sucesor en el árbol. Los métodos de recorrido (InOrder, PreOrder, PostOrder) permiten validar la correcta disposición de los nodos, y el menú de TestAVL da acceso interactivo a todas estas funcionalidades.



iii. **EJERCICIO 3:** : Implementar un método para graficar el árbol AVL resultante, mostrando todos sus nodos, sus aristas izquierda y derecha utilizando clases y métodos genéricos, utilizar la librería Graph Stream o similar.

Se integra la librería GraphStream (versión 1.3) para generar un SingleGraph donde cada nodo del AVL se añade automáticamente usando su clave como ID

y etiqueta. El método recursivo addToGraph pinta aristas dirigidas etiquetadas "L" o "R" según sean hijos izquierdos o derechos. Un sencillo stylesheet en Swing define color, tamaño y flechas, y graph.display() abre la ventana interactiva que permite visualizar en tiempo real la estructura del árbol balanceado.



II. SOLUCIÓN DEL CUESTIONARIO

a. ¿Cuáles fueron las dificultades que encontraste al desarrollar los ejercicios propuestos? por ejemplo, poca documentación, complejidad del lenguaje, etc.

- **Gestión del factor de equilibrio:** Mantener el cálculo correcto de bf tras cada inserción y eliminación resultó complejo, sobre todo al actualizar alturas y decidir si aplicar rotaciones simples o dobles.

- **Rotaciones dobles:** Encontrar documentación clara y ejemplos precisos para los casos izquierdo-derecha y derecha-izquierda fue difícil, lo que

obligó a revisar varios artículos y libros para asegurar una implementación libre de errores.

- **Código genérico en Java:** Manejar la interfaz Comparable<T> y garantizar que el árbol acepte cualquier tipo comparable añadió cierta verbosidad al código, y hubo que prestar atención a las conversiones de tipos y restricciones de genéricos.

- **Pruebas exhaustivas:** Verificar que tras secuencias largas de inserciones y eliminaciones el árbol permaneciera balanceado requirió múltiples casos de prueba, lo que complicó la fase de debugging.

- **Integración de GraphStream:** La API de GraphStream tiene su propia curva de aprendizaje: comprender cómo funciona su modelo de nodos y aristas, configurar el layout y aplicar estilos CSS en Swing llevó tiempo de experimentación.

b. ¿Explique cómo es el algoritmo que implementó para obtener el AVL con la librería Graph Stream? Recuerde que puede agregar operaciones sobre la clase BST.

El método visualize() crea un SingleGraph de GraphStream configurado en modo Swing y define un stylesheet básico para nodos y flechas. A continuación invoca recursivamente addToGraph(node, graph), que para cada nodo AVL añade (o recupera) un vértice identificado por su clave y lo etiqueta con su valor; luego crea aristas dirigidas etiquetadas “L” y “R” hacia los hijos izquierdo y derecho, respectivamente. Esta llamada recursiva garantiza que todo el subárbol quede representado en el grafo. Finalmente, graph.display() abre una ventana interactiva que muestra la topología jerárquica del AVL. Opcionalmente, antes o después de la visualización se pueden invocar operaciones propias de un BST—como recorrido por niveles, conteo de nodos o cálculo de profundidad—inyectando esa información en la ventana o imprimiéndola en consola para complementar la gráfica.

III. CONCLUSIONES

- El trabajo permitió afianzar el conocimiento sobre árboles balanceados, en particular la gestión del factor de equilibrio y las rotaciones necesarias (simples y dobles) tras cada inserción o eliminación para garantizar que la altura del árbol se mantenga en O(logn).

- Se comprobó la versatilidad de las clases genéricas (NodeAVL<T>, AVLTree<T>) para operar con cualquier tipo que implemente Comparable<T>, lo cual facilita la reutilización del código y mantiene la seguridad de tipos sin sacrificar rendimiento ni claridad.
- En el Ejercicio 2 se integraron operaciones propias de un BST —búsqueda, mínimo, máximo, predecesor y sucesor— junto con recorridos (InOrder, PreOrder, PostOrder) y las funciones de destrucción y verificación de vacío, consolidando una API completa y coherente.
- La incorporación de GraphStream en el Ejercicio 3 ofreció una representación gráfica e interactiva del árbol, facilitando la comprensión de la topología y la comprobación visual del balance. Esto demuestra cómo herramientas externas pueden integrarse para mejorar la usabilidad y el análisis de estructuras de datos.
- Los principales desafíos se centraron en depurar las rotaciones dobles y en configurar adecuadamente el layout y estilo de GraphStream. Sin embargo, estas dificultades se tradujeron en un aprendizaje profundo sobre balanceo de árboles y en experiencia práctica con bibliotecas de visualización en Java.

REFERENCIAS Y BIBLIOGRAFÍA

- Deitel, P. J., & Deitel, H. M. (2014). *Java: Cómo programar* (10.ª ed.). Pearson Educación.
- Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2013). *Estructuras de datos y algoritmos en Java* (6.ª ed.). Wiley.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms* (3rd ed.). MIT Press.
- GraphStream Team. (2023). *GraphStream – A dynamic graph library*. https://graphstream-project.org/
- Oracle. (2024). *The Java™ tutorials*. https://docs.oracle.com/javase/tutorial/
- Stack Overflow. (n.d.). *Various discussions and solutions for configuring external libraries in Java projects*. https://stackoverflow.com