Universidad Tecmilenio

Actividad 4

Estructuras De Datos

Profesor:
Adalberto Emmanuel Rojas Perea

Alumno:
Zaid Castillo Hermosillo

Se implementó un árbol binario en Java capaz de guardar Empleados con sus respectivos IDs y nombres. Es posible la inserción, eliminación y búsqueda de la información además de mostrar los distintos recorridos que se pueden tomar para organizarse.

Clases creadas:

Empleado

Resulta la clase elemental del programa ya que almacena los datos a manipular, se establece un ID y un nombre como atributo de cada empleado.

```
public class Empleado implements Comparable<Empleado> {
    private int ID;
    private String nombre;
    public Empleado(int ID, String nombre){
        this.ID= ID;
        this.nombre= nombre;
    }

    public Empleado(int ID){
        this.ID= ID;
    }

    public int getID() {
        return ID;
    }
    public String getNombre() {
        return nombre;
    }
```

Así como sus getters y setters, y se sobreescribe el método “toString()” para así mostrar la información del objeto instanciado como Empleado de manera organizada sin que muestre solamente la posición en memoria del objeto.

```
@Override
public String toString() {
    return "Empleado{" +
            "ID=" + ID +
            ", nombre='" + nombre + '\'' +
            '}';
}
```

Se agregó también un método para comparar los IDs de otros empleados y así facilitar la busqueda y eliminación de empleados.

```
// Metodo para comparar los IDs de los Empleados que se instancien
@Override
public int compareTo(Empleado o) {
    if(this.ID>o.getID()) {
        return 1;
    } else if (this.ID<o.getID()) {
        return -1;
    }
    return 0;
}
```

Nodo

Clase que alberga la clase Empleado y establece los punteros para organizar el árbol binario. Cuenta con los mismos métodos mencionados en la clase anterior.

```
public class Nodo {
    private Empleado empleado;
    private Nodo hijoIzquierdo;
    private Nodo hijoDerecho;

    public Nodo (Empleado empleado) {
        this.empleado = empleado;
        this.hijoIzquierdo=null;
        this.hijoDerecho=null;
    }

    public Empleado getEmpleado() {
        return this.empleado;
    }
    …

    @Override
    public String toString() {
        return "Empleado{" +
                "ID=" + this.empleado.getID() +
                ", nombre='" + this.empleado.getNombre() + '\'' +
                '}';
    }
}
```

Árbol Binario

Clase para instanciar el árbol a manejar. Se incluyen los métodos necesarios para las funcionalidades requeridas haciendo uso de las clases previas.

```
public class ArbolBinario {
    Nodo raiz = null;
    private int numElementos = 0;
```

Se especifica la variable raíz ya que se usa como base para muchos de los métodos, y una variable para ir contando los elementos que se van agregando/eliminando.

Método insertar elemento: Inicializa el empleado a colocar como un nodo y checa si ya hay raíz, si no, se inserta ahí.Después, se checa si tiene hijos y se compara el valor de su ID, si alguno es null, se inserta dependiendo de su tamaño (si es mayor de lado derecho y si es menor de lado izquierdo), y si ninguno es null se siguen buscando los lugares disponibles. Después de ser colocado el nodo, se especficia su posición en cuanto a los demás (e.g. Empleado con ID 23 insertado como hijo derecho de empleado con ID 10).

```
public void insertarElemento(Empleado empleado){
    Nodo nuevoNodo = new Nodo(empleado);
    if (this.raiz == null) {
       this.raiz = nuevoNodo;
        JOptionPane.showMessageDialog(null, "Se puso la raíz como: " +
empleado, "Localización de empleado", JOptionPane.INFORMATION_MESSAGE);
    } else {
```

```
        Nodo temp = raiz;
        while(temp != null) {
            if (temp.getHijoDerecho() == null && temp.getHijoIzquierdo() ==
null) {
                if (empleado.compareTo(temp.getEmpleado()) == 1) {
                    JOptionPane.showMessageDialog(null, empleado + "
insertado como hijo derecho de: " + temp, "Localización de empleado",
JOptionPane.INFORMATION_MESSAGE);
                    temp.setHijoDerecho(nuevoNodo);
                    temp = null;
                } else {
                    JOptionPane.showMessageDialog(null, empleado + "
insertado como hijo izquierdo de: " + temp, "Localización de empleado",
JOptionPane.INFORMATION_MESSAGE);
                    temp.setHijoIzquierdo(nuevoNodo);
                    temp = null;
                }
            } else if (empleado.compareTo(temp.getEmpleado()) == 1 &&
temp.getHijoDerecho() == null) {
                JOptionPane.showMessageDialog(null, empleado + " insertado
como hijo derecho de: " + temp, "Localización de empleado",
JOptionPane.INFORMATION_MESSAGE);
                temp.setHijoDerecho(nuevoNodo);
                temp = null;
            } else if (empleado.compareTo(temp.getEmpleado()) == -1 &&
temp.getHijoIzquierdo() == null) {
                JOptionPane.showMessageDialog(null, empleado + " insertado
como hijo izquierdo de: " + temp, "Localización de empleado",
JOptionPane.INFORMATION_MESSAGE);
                temp.setHijoIzquierdo(nuevoNodo);
                temp = null;
            } else {
                if (empleado.compareTo(temp.getEmpleado()) == 1) {
                    temp = temp.getHijoDerecho();
                } else {
                    temp = temp.getHijoIzquierdo();
                }
            }
        }
    }
    numElementos++;
}
```

Método para buscar un empleado usando como parametro el ID: Se crea un empleado nuevo usando solamente el parametro de id, después se comparan los ids, de los elementos en el árbol y si se encuentra se devuelve el empleado con el mismo ID.

```
public Empleado buscarEmpleado (int id) {
    Empleado valor = new Empleado(id);
    Nodo temp = raiz;
    Nodo encontrado = null;

    while (temp!=null){
        if (valor.compareTo(temp.getEmpleado())==0){
            encontrado = temp;
            temp = null;
        } else if(temp.getHijoDerecho() == null && temp.getHijoIzquierdo() ==
```

```
null) {
            temp = null;
        } else if(valor.compareTo(temp.getEmpleado()) == 1 &&
temp.getHijoDerecho() != null) {
            temp = temp.getHijoDerecho();
        } else if(valor.compareTo(temp.getEmpleado()) == -1 &&
temp.getHijoIzquierdo() != null) {
            temp = temp.getHijoIzquierdo();
        } else {
            temp = null;
        }
    }
    if (encontrado != null) {
        return encontrado.getEmpleado();
    } else {
        return null;
    }
}
```

Método para recorrer el árbol en inorden: Se recorre la rama izquierda, después la raíz y por último el lado derecho usando la recursión.

```
public void inorden(){
    String recorrido = "Recorrido Inorden (Izquierdo - Raíz - Derecho):\n";
    recorrido += inorden(this.raiz);
    recorrido += "Número de empleados registrados: " +this.size();
    JOptionPane.showMessageDialog(null, recorrido, "Recorrido Inorden",
JOptionPane.INFORMATION_MESSAGE);
}
public String inorden (Nodo r) {
    if (r == null) {
        return "";
    }
    return inorden(r.getHijoIzquierdo()) +
            r + "\n" +
            inorden(r.getHijoDerecho());
}
```

Método para recorrer el árbol en preorden: Se empieza por la raíz, después el lado izquierdo y por último el lado derecho usando la recursión.

```
public void preorden(){
    String recorrido = "Recorrido Preorden (Raíz - Izquierdo - Derecho):\n";
    recorrido += preorden(this.raiz);
    recorrido += "Número de empleados registrados: " +this.size();
    JOptionPane.showMessageDialog(null, recorrido, "Recorrido Preorden",
JOptionPane.INFORMATION_MESSAGE);
}
public String preorden (Nodo r) {
    if (r== null) {
        return "";
    }
    return r + "\n" +
            preorden(r.getHijoIzquierdo()) +
            preorden(r.getHijoDerecho());
    }
```

Método para recorrer el árbol en postorden: Se recorre la rama izquierda, después la derecha y por último la raíz.

```
public void postorden() {
    String recorrido = "Recorrido Postorden (Izquierdo - Derecho - Raíz):\n";
    recorrido += postorden(this.raiz);
    recorrido += "Número de empleados registrados: " +this.size();
    JOptionPane.showMessageDialog(null, recorrido, "Recorrido Postorden",
JOptionPane.INFORMATION_MESSAGE);
}
public String postorden (Nodo r) {
    if (r == null) {
        return "";
    }
    return postorden(r.getHijoIzquierdo()) +
            postorden(r.getHijoDerecho()) +
            r + "\n";
}
```

Método para eliminar un empleado del árbol. Se checa si hay raíz, después se toma el empleado que se está buscando y se compara con los elementos que tenga el árbol dependiendo de sus posiciones. Después, se reorganizan las posiciones basandose en el elemento eliminado.

```
public boolean eliminarElemento(Empleado empleado) {
    if (raiz == null) return false;
    raiz = eliminarElemento(raiz, empleado);
    numElementos--;
    return true;
}
private Nodo eliminarElemento(Nodo actual, Empleado empleado) {
    if (actual == null) return null;
    int cmp = empleado.compareTo(actual.getEmpleado());
    if (cmp < 0) {
        actual.setHijoIzquierdo(eliminarElemento(actual.getHijoIzquierdo(),
empleado));
    } else if (cmp > 0) {
        actual.setHijoDerecho(eliminarElemento(actual.getHijoDerecho(),
empleado));
    } else {
        // Caso 1: sin hijos
        if (actual.getHijoIzquierdo() == null && actual.getHijoDerecho() ==
null) {
            return null;
        }
        // Caso 2: una hoja
        if (actual.getHijoIzquierdo() == null) {
            return actual.getHijoDerecho();
        } else if (actual.getHijoDerecho() == null) {
            return actual.getHijoIzquierdo();
        }
        // Caso 3: dos hojas
        Nodo sucesor = encontrarMin(actual.getHijoDerecho());
        actual.setEmpleado(sucesor.getEmpleado());
        actual.setHijoDerecho(eliminarElemento(actual.getHijoDerecho(),
sucesor.getEmpleado()));
    }
```

```
    return actual;
}
private Nodo encontrarMin(Nodo nodo) {
    while (nodo.getHijoIzquierdo() != null) {
        nodo = nodo.getHijoIzquierdo();
    }
    return nodo;
}
```

Método para acceder al atributo contador de empleados ingresados.

```
public int size() {
    return this.numElementos;
}
```

Método para saber si el árbol está vacío checando si la raíz es null.

```
public boolean isEmpty(){
    return raiz == null;
}
```

Main

En esta clase se inicializan todas las previas y se establece un menú para poder usar los métodos requeridos. Se utiliza Java Swing para proveer una GUI tanto para la captura de datos como para imprimir los resultados.

```
do {
    try {
        String input = JOptionPane.showInputDialog(
                null,
                "1. Agregar un Empleado\n"
                        + "2. Recorrer el árbol (preorden)\n"
                        + "3. Recorrer el árbol (inorden)\n"
                        + "4. Recorrer el árbol (postorden)\n"
                        + "5. Buscar empleado por ID\n"
                        + "6. Eliminar empleado por ID\n"
                        + "7. Salir\n\n"
                        + "Elige una opción...",
                "Árboles Binarios",
                JOptionPane.QUESTION_MESSAGE
        );
        if (input == null) {
            opcion = 7;
        } else {
            opcion = Integer.parseInt(input);
        }

        switch (opcion) {
            case 1 -> {
                int id = Integer.parseInt(
                        JOptionPane.showInputDialog(
                                null, "Ingresa el ID (número)...",
                                "Agregando Empleado",
JOptionPane.QUESTION_MESSAGE
                        )
                );
                String nombre = JOptionPane.showInputDialog(
                        null, "Ingresa el nombre del empleado",
```

```
                    "Agregando Empleado", JOptionPane.QUESTION_MESSAGE
            );
            Empleado nuevo = new Empleado(id, nombre);
            arbol.insertarElemento(nuevo);
        }
        ...
```