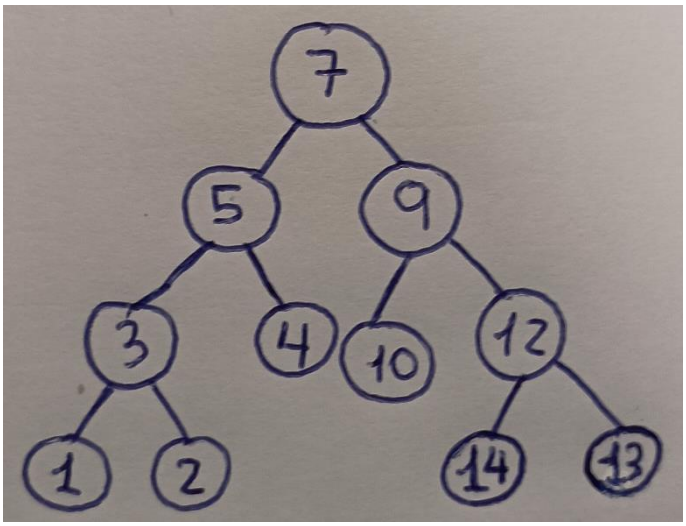
MEMORIA DE LA PRÁCTICA 3: TAD ABB 2023



28 MAYO

UNIVERSIDAD DE VALLADOLID
Creado por: Raúl Colindres de Lucas y Alfredo Sobrados González
Nº de grupo de las prácticas: 1
Turno de laboratorio: Miércoles de 18:00h a 20:00h
Fecha de entrega: Lunes 29 de mayo de 2023, 22h

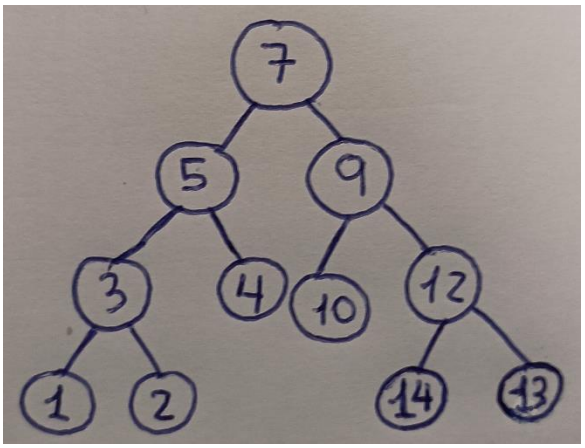
CAMPUS PÚBLICO
MARÍA ZAMBRANO
SEGOVIA

Especificación lógica completa

Especificación lógica completa de los TAD ABB y ABBEnteros diseñados

Especificación Lógica del TAD ABB

1.-Definición: el TAD a diseñar es un ABB (árbol binario de búsqueda), los ABB se componen de nodos los cuales pueden tener dos hijos como mucho, el nodo raíz es por donde se accede a la estructura de datos y este tiene un subarbol a la izquierda con valores mas pequeños que él y tiene un subarbol a la derecha con valores mas grandes que él.



2.-Elementos: los elementos que almacena este TAD son claves enteras.
3.-Tipo de organización: es una estructura de datos jerárquica.
4.-Dominio de los elementos del TAD: el dominio está compuesto por todos los valores enteros que se puedan representar en Java que es el lenguaje para implementar el TAD.
5.-Operaciones Básicas:
Nombre: recuperar
Descripción: recupera el elemento del árbol binario de búsqueda que se pasa por parámetro.
Datos de entrada: el elemento que se quiere recuperar.
Datos de salida: el elemento recuperado del árbol binario de búsqueda, si se recupera si no se lanza una excepción de usuario.
Precondiciones: el árbol binario de búsqueda de enteros debe existir y contener al menos un nodo.
Postcondiciones: ninguna.

Nombre: recuperarMin
Descripción: recupera el elemento más grande del árbol binario de búsqueda.
Datos de entrada: ninguno.
Datos de salida: el elemento más grande del árbol binario de búsqueda.
Precondiciones: el árbol binario de búsqueda de enteros debe existir y contener al menos un nodo.
Postcondiciones: ninguna.

Nombre: recuperarMax
Descripción: recupera el elemento más grande del árbol binario de búsqueda.
Datos de entrada: ninguno.
Datos de salida: el elemento más grande del árbol binario de búsqueda.
Precondiciones: el árbol binario de búsqueda de enteros debe existir y contener al menos un nodo.
Postcondiciones: ninguna.

Nombre: insertarSinDuplicados
Descripción: inserta elementos en el ABB que no estén ya insertados anteriormente.
Datos de entrada: el elemento a insertar en el ABB.
Datos de salida: ninguno
Precondiciones: ninguna
Postcondiciones: que los elementos insertados por la operación no estén duplicados.

Nombre: insertarConDuplicados
Descripción: inserta elementos en el ABB estén o no estén antes en el ABB.
Datos de entrada: el elemento a insertar en el ABB.
Datos de salida: ninguno
Precondiciones: ninguna
Postcondiciones: ninguna.

Nombre: insertarEquilibrado
Descripción: inserta los elementos en el ABB de manera equilibrada o balanceada.
Datos de entrada: el vector con los elementos a insertar.
Datos de salida: ninguno.
Precondiciones: ninguna.
Postcondiciones: ninguna.

Nombre: eliminar
Descripción: elimina el elemento que se le pase por parámetro dentro del ABB.
Datos de entrada: el elemento que se quiere eliminar dentro del ABB.
Datos de salida: ninguno.
Precondiciones: que haya por lo menos un elemento o varios dentro del ABB
Postcondiciones: ninguna.

Nombre: eliminarMin
Descripción: elimina el elemento más pequeño dentro del ABB
Datos de entrada: ninguno
Datos de salida: el elemento eliminado, que es el más pequeño del ABB
Precondiciones: que el ABB tenga uno o más elementos.
Postcondiciones: ninguna.

Nombre: tamanyo
Descripción: devuelve el tamaño del ABB.
Datos de entrada: ninguno.
Datos de salida: el tamaño del ABB.
Precondiciones: ninguna.
Postcondiciones: ninguna.

Nombre: altura
Descripción: devuelve la altura del ABB.
Datos de entrada: ninguno.
Datos de salida: la altura del ABB.
Precondiciones: ninguna.
Postcondiciones: no se modifican los nodos ni la estructura del árbol.

Nombre: esVacio
Descripción: comprueba si el ABB está vacío o no.
Datos de entrada: ninguno
Datos de salida: true si está vacío o false si no está vacío.
Precondiciones: ninguna.
Postcondiciones: no se modifican los nodos ni la estructura del árbol.

Nombre: toStringPreOrden
Descripción: devuelve un String con toda la información del ABB en este orden: raíz-IZQ-DER.
Datos de entrada: ninguno.
Datos de salida: el String con la información.
Precondiciones: el árbol binario de búsqueda de enteros debe existir y contener al menos un nodo.
Postcondiciones: no se modifican los nodos ni la estructura del árbol.

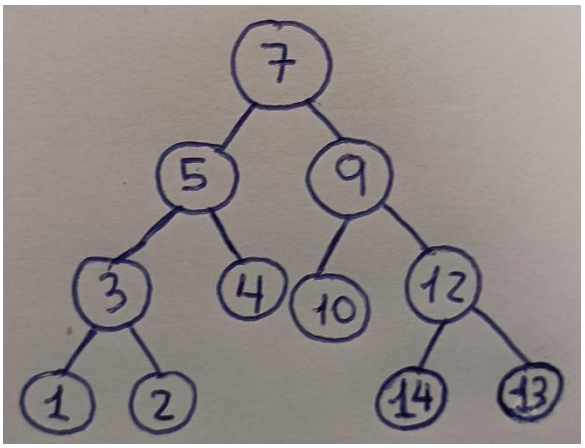
Nombre: toStringPostOrden
Descripción: devuelve un String con toda la información del ABB en este orden: IZQ-DER-raíz.
Datos de entrada: ninguno.
Datos de salida: el String con la información.
Precondiciones: el árbol binario de búsqueda de enteros debe existir y contener al menos un nodo.
Postcondiciones: no se modifican los nodos ni la estructura del árbol.

Nombre: toStringInOrden
Descripción: devuelve un String con toda la información del ABB en este orden: IZQ-raíz-DER.
Datos de entrada: ninguno.
Datos de salida: el String con la información.
Precondiciones: el árbol binario de búsqueda de enteros debe existir y contener al menos un nodo.
Postcondiciones: no se modifican los nodos ni la estructura del árbol.

Nombre: toStringPorNiveles
Descripción: devuelve un String con toda la información del ABB empezando por la raíz y yendo desde el primer nivel hasta el último, te muestra los elementos en orden en un String.
Datos de entrada: ninguno.
Datos de salida: el String con la información.
Precondiciones: el árbol binario de búsqueda de enteros debe existir y contener al menos un nodo.
Postcondiciones: no se modifican los nodos ni la estructura del árbol.

Especificación Lógica del TAD ABBEnteros

1.-Definición: el TAD a diseñar es un ABB (árbol binario de búsqueda), los ABB se componen de nodos los cuales pueden tener dos hijos como mucho, el nodo raíz es por donde se accede a la estructura de datos y este tiene un subarbol a la izquierda con valores mas pequeños que él y tiene un subarbol a la derecha con valores mas grandes que él.



2.-Elementos: los elementos que almacena este TAD son claves enteras unicamente.
3.-Tipo de organización: es una estructura de datos jerárquica.
4.-Dominio de los elementos del TAD: el dominio está compuesto por todos los valores enteros que se puedan representar en Java que es el lenguaje para implementar el TAD.
5.-Operaciones Básicas:
Nombre: recuperar
Descripción: recupera el entero del árbol binario de búsqueda de enteros que se pasa por parámetro.
Datos de entrada: el entero que se quiere recuperar.
Datos de salida: el entero recuperado del árbol binario de búsqueda de enteros, si se recupera si no se lanza una excepción de usuario.
Precondiciones: ninguna.
Postcondiciones: no se modifican los nodos ni la estructura del árbol.

Nombre: recuperarMin
Descripción: recupera el elemento más grande del árbol binario de búsqueda de enteros.
Datos de entrada: ninguno.
Datos de salida: el entero más grande del árbol binario de búsqueda de enteros.
Precondiciones: El árbol binario de búsqueda de enteros debe existir y contener al menos un nodo.
Postcondiciones: no se modifican los nodos ni la estructura del árbol.

Nombre: recuperarMax
Descripción: recupera el elemento más grande del árbol binario de búsqueda de enteros.
Datos de entrada: ninguno.
Datos de salida: el entero más grande del árbol binario de búsqueda de enteros.
Precondiciones: El árbol binario de búsqueda de enteros debe existir y contener al menos un nodo.
Postcondiciones: no se modifican los nodos ni la estructura del árbol.

Nombre: insertarSinDuplicados
Descripción: inserta elementos en el ABBEnteros que no estén ya insertados anteriormente.
Datos de entrada: el entero a insertar en el ABBEnteros.
Datos de salida: el ABB con el entero insertado.
Precondiciones: ninguna
Postcondiciones: que se haya insertado el elemento correctamente del ABBEnteros y no este duplicado.

Nombre: insertarConDuplicados
Descripción: inserta enteros en el ABBEnteros estén o no estén antes en el ABBEnteros.
Datos de entrada: el entero a insertar en el ABBEnteros.
Datos de salida: el ABB con el entero insertado.
Precondiciones: ninguna.
Postcondiciones: que se haya insertado el elemento correctamente del ABBEnteros.

Nombre: insertarEquilibrado
Descripción: inserta los enteros en el ABBEnteros de manera equilibrada o balanceada.
Datos de entrada: el vector con los enteros a insertar.
Datos de salida: el ABB con el vector de enteros insertado.
Precondiciones: ninguna.
Postcondiciones: que se haya insertado el elemento correctamente del ABBEnteros.

Nombre: eliminar
Descripción: elimina el elemento que se le pase por parámetro dentro del ABBEnteros.
Datos de entrada: el elemento que se quiere eliminar dentro del ABBEnteros.
Datos de salida: el ABB.
Precondiciones: que haya por lo menos un elemento o varios dentro del ABBEnteros
Postcondiciones: que se haya eliminado el elemento correctamente del ABBEnteros.

Nombre: eliminarMin
Descripción: elimina el entero más pequeño dentro del ABBEnteros
Datos de entrada: el ABB.
Datos de salida: el entero eliminado, que es el más pequeño del ABBEnteros
Precondiciones: que el ABB tenga uno o más enteros.
Postcondiciones: no se modifican los nodos ni la estructura del árbol.

Nombre: tamanyo
Descripción: devuelve el tamaño del ABBEnteros.
Datos de entrada: el ABB.
Datos de salida: el tamaño del ABBEnteros.
Precondiciones: ninguna.
Postcondiciones: no se modifican los nodos ni la estructura del árbol.

Nombre: altura
Descripción: devuelve la altura del ABBEnteros.
Datos de entrada: el ABB.
Datos de salida: la altura del ABBEnteros.
Precondiciones: El árbol binario de búsqueda de enteros debe existir y contener al menos un nodo.
Postcondiciones: no se modifican los nodos ni la estructura del árbol.

Nombre: esVacio
Descripción: comprueba si el ABBEnteros está vacío o no.
Datos de entrada: el ABB.
Datos de salida: true si está vacío o false si no está vacío.
Precondiciones: El árbol binario de búsqueda de enteros debe existir y contener al menos un nodo.
Postcondiciones: no se modifican los nodos ni la estructura del árbol.

Nombre: toStringPreOrden
Descripción: devuelve un String con todos los enteros del ABBEnteros en este orden: raíz-IZQ-DER.
Datos de entrada: el ABB.
Datos de salida: el String con los enteros.
Precondiciones: El árbol binario de búsqueda de enteros debe existir y contener al menos un nodo.
Postcondiciones: no se modifican los nodos ni la estructura del árbol.

Nombre: toStringPostOrden
Descripción: devuelve un String con todos los enteros del ABBEnteros en este orden: IZQ-DER-raíz.
Datos de entrada: el ABB.
Datos de salida: el String con los enteros.
Precondiciones: El árbol binario de búsqueda de enteros debe existir y contener al menos un nodo.
Postcondiciones: no se modifican los nodos ni la estructura del árbol.

Nombre: toStringInOrden
Descripción: devuelve un String con todos los enteros del ABBEnteros en este orden: IZQ- raíz-DER.
Datos de entrada: el ABB.
Datos de salida: el String con los enteros.
Precondiciones: El árbol binario de búsqueda de enteros debe existir y contener al menos un nodo.
Postcondiciones: no se modifican los nodos ni la estructura del árbol.

Nombre: toStringInOrdenInverso
Descripción: devuelve un String con todos los enteros del ABBEnteros en este orden: DER- raíz- IZQ.
Datos de entrada: el ABB.
Datos de salida: el String con los enteros.
Precondiciones: El árbol binario de búsqueda de enteros debe existir y contener al menos un nodo.
Postcondiciones: no se modifican los nodos ni la estructura del árbol.

Nombre: toStringPorNiveles
Descripción: devuelve un String con todos los enteros del ABBEnteros empezando por la hoja que está más a la izquierda, yendo desde el último nivel hasta el primero y yendo de izquierda a derecha en cada nivel.
Datos de entrada: el ABB.
Datos de salida: el String con los enteros.
Precondiciones: El árbol binario de búsqueda de enteros debe existir y contener al menos un nodo.
Postcondiciones: no se modifican los nodos ni la estructura del árbol.

Nombre: toStringPorNivelesInverso
Descripción: devuelve un String con todos los enteros del ABBEnteros empezando por la raíz, yendo desde el primer nivel hasta el último y yendo de derecha a izquierda en cada nivel.
Datos de entrada: el ABB.
Datos de salida: el String con los enteros.
Precondiciones: El árbol binario de búsqueda de enteros debe existir y contener al menos un nodo.
Postcondiciones: no se modifican los nodos ni la estructura del árbol.

Nombre: transformarABBEnteros
Descripción: esta operación transforma el ABB de enteros de manera que las claves enteras positivas que posea el ABB de enteros se transforman en claves enteras negativas.
Datos de entrada: el ABB.
Datos de salida: el ABB transformado.
Precondiciones: que el ABB de enteros no este vacío.
Postcondiciones: que el ABB de enteros tenga todas sus claves enteras negativas.

Nombre: sumarClavesMinNumDado
Descripción: suma las claves menores del ABB menores a un número dado, que es el que introduce por teclado el usuario.
Datos de entrada: el ABB.
Datos de salida: ninguno.
Precondiciones: El árbol binario de búsqueda de enteros debe existir y contener al menos un nodo.
Postcondiciones: ninguna.

Nombre: calcularAntecesorNodo
Descripción: calcula el antecesor de un nodo en el árbol binario de búsqueda de enteros. Se solicita al usuario que ingrese la clave del nodo. Para calcular el antecesor primero busca un nodo con la clave introducida en el árbol binario de búsqueda de enteros a partir del nodo raíz. Y luego encuentra el antecesor del nodo introducido por teclado en el árbol a partir del nodo raíz.
Datos de entrada: Clave del nodo (valor introducido por el usuario).
Datos de salida: ninguno
Precondiciones: El árbol binario de búsqueda de enteros debe existir y contener al menos un nodo.
Postcondiciones: No se modifican los nodos ni la estructura del árbol. Si se encuentra un nodo con la clave introducida, se determina y muestra su antecesor. Si no se encuentra un nodo con la clave introducida, se muestra un mensaje indicando que el nodo no existe en el árbol.

Diseño de los métodos recursivos

Diseño de los métodos recursivos (Análisis del problema / Identificación de caso/s base / Identificación de caso/s recursivo/s)

Método recursivo toStringInOrdenInverso:

- Análisis del problema: El objetivo de este método es generar una representación en cadena de un árbol en orden inverso, es decir, recorriendo primero el subárbol derecho, luego la raíz y finalmente el subárbol izquierdo.
- Identificación de caso/s base:
    - Si el nodo actual es nulo, se retorna una cadena vacía.
    - Si el nodo actual no tiene hijos (tanto izquierdo como derecho son nulos), se retorna la representación en cadena del valor del nodo actual.
- Identificación de caso/s recursivo/s:
  Para cada nodo, se realizará la siguiente secuencia de pasos:
    1. Llamar recursivamente al método toStringInOrdenInverso para el subárbol derecho.
    2. Concatenar la representación en cadena del valor del nodo actual.
    3. Concatenar el resultado de llamar recursivamente al método toStringInOrdenInverso para el subárbol izquierdo.
    4. Retornar la cadena resultante.
- Descripción: devuelve un String con todos los enteros del ABBEnteros en este orden: DER- raíz- IZQ.

Método recursivo toStringPorNivelesInverso:

- Análisis del problema: Este método tiene como objetivo generar una representación en cadena de un árbol siguiendo un recorrido por niveles en orden inverso, es decir, se recorren los nodos de mayor profundidad primero.
- Identificación de caso/s base:
    - Si el nodo actual es nulo, se retorna una cadena vacía.
    - Si el nodo actual no tiene hijos (tanto izquierdo como derecho son nulos), se retorna la representación en cadena del valor del nodo actual.
- Identificación de caso/s recursivo/s:
  Para cada nivel del árbol, se realizará la siguiente secuencia de pasos:
    1. Se obtiene la representación en cadena de todos los nodos del nivel actual (recorriendo de izquierda a derecha).

2. Luego, se realiza una llamada recursiva al método toStringPorNivelesInverso para el subárbol izquierdo.
3. Concatenar la cadena resultante del paso anterior con la representación en cadena de los nodos del nivel actual.
4. Retornar la cadena resultante.

- Descripción: devuelve un String con todos los enteros del ABBEnteros empezando por la raíz, yendo desde el primer nivel hasta el último y yendo de derecha a izquierda en cada nivel.

Método recursivo buscarNodo:

- Análisis del problema: recorres el árbol binario de búsqueda hasta encontrar la clave buscada si el árbol esta vacío o el nodo buscado es la raíz, devuelve la raíz si la clave es más pequeña que la raíz buscamos en la izquierda y si es más grande por la derecha.
- Identificación de caso/s base: las llamadas recursivas paran cuando se encuentra la clave buscada o cuando se ha recorrido todo el árbol y no se ha encontrado la clave buscada.
- Identificación de caso/s recursivo/s: si la clave es más pequeña que la raíz se hacen llamadas recursivas por la izquierda para recorrer el árbol hasta encontrar la clave buscada y si es más grande hace lo mismo, pero por la derecha de la raíz.
- Descripción: si la raíz del árbol es nula o el dato de la raíz es igual a la clave entonces de devuelve la raíz. Si la clave es menor que el dato de la raíz entonces se busca la clave por el subárbol izquierdo y si no se busca por el subárbol derecho.

Método recursivo encontrarAntecesor:

- Análisis del problema: si el árbol está vacío se devuelve null si el puntero a la derecha o la izquierda es igual al nodo del que estamos buscando su antecesor entonces su antecesor será la raíz del árbol y si la clave del nodo del que queremos saber su antecesor es más pequeña que la raíz se buscara recursivamente hasta dar con el nodo y su antecesor e igual con las claves que son más grandes que la raíz, pero se buscara por la derecha de la raíz del árbol.
- Identificación de caso/s base: las llamadas paran cuando se encuentran la clave buscada y su antecesor o cuando se ha recorrido todo el árbol y no se han encontrado la clave y su antecesor.
- Identificación de caso/s recursivo/s: si la clave es más pequeña que la raíz se hacen llamadas recursivas por la izquierda para recorrer el árbol hasta encontrar

la clave y su antecesor y si es más grande hace lo mismo, pero por la derecha de la raíz.

- Descripción: si la raíz del árbol es nula o el nodo del que queremos encontrar el antecesor es nulo, se devuelve null. Si el nodo izquierdo o el nodo derecho de la raíz es igual al nodo, entonces su antecesor es la raíz. Si el dato del nodo es menor que el dato de la raíz entonces se busca el antecesor del nodo por el subárbol izquierdo y si no se busca por el subárbol derecho.

Método recursivo transformarABBEnteros:

- Análisis del problema: si el árbol está vacío se lanza una excepción y no se hace nada si el árbol tiene contenido, entonces se lanza la primera llamada recursiva y se van haciendo llamadas recursivas para recorrer todo el árbol de manera que las claves positivas que tiene el árbol se transformen en claves negativas.
- Identificación de caso/s base: el caso base es cuando se llegue a los nodos hoja del árbol y se haya transformado todo el ABB.
- Identificación de caso/s recursivo/s: si no se encuentren referencias a null por la izquierda y por la derecha del nodo actual se siguen haciendo llamadas recursivas y se siguen transformando las claves si su referencia no es nula.
- Descripción: transforma el elemento del nodo actual a su correspondiente valor negativo, y se realiza la transformación recursivamente en los nodos hijos.

Método recursivo sumarClavesMinNumDado:

- Análisis del problema: el usuario introduce una clave por teclado y en función de esa clave recorre el ABB de manera recursiva sumando las claves correspondientes.
- Identificación de caso/s base: si la raíz es nula, es decir, el ABB está vacío, se devuelve 0, si no se recorre el árbol en función del número dado y se termina cuando se haya recorrido todo y la suma este hecha.
- Identificación de caso/s recursivo/s: en función del número dado se ira recorriendo el ABB y sumando sus claves con llamadas recursivas hasta llegar a los nodos hoja.
- Descripción: suma las claves menores del ABB menores a un número dado, que es el que introduce por teclado el usuario.

Método recursivo mostrarHojasPosImpares:

- Análisis del problema: si el árbol está vacío no hace nada, si el nodo actual es una hoja y su posición es impar se muestra un mensaje en pantalla en el que se pone la posición y clave entera de la hoja (nodo actual) y se recorren los hijos izquiero y derecho en caso de que el nodo actual no sea nodo hoja en posición impar.
- Identificación de caso/s base: cuando se ha recorrido todo el ABB y ya no hay más nodos hoja en posiciones impares en el ABB.
- Identificación de caso/s recursivo/s: cuando el nodo actual tiene hijo/s izquierdo y/o derecho se hacen llamadas recursivas.
- Descripción: muestra las hojas de los subárbol izquierdo y derecho en posiciones impares

Listado del código fuente

Listado del código fuente de la aplicación desarrollada

Clase MyInput:

```
package EntradaSalida;

import java.io.*;
import java.util.ArrayList;
/**
 * Esta clase contiene los metodos y atributos de los procesos E/S que se
precisan en este programa.
 * @author Raul Colindres y Alfredo Sobrados
 */
public class MyInput {
    /**
     * Metodo que lee una cadena de caracteres desde el teclado.
     * @return string devuelve el string leído a traves de la pantalla
     */
public static String readString() {
    BufferedReader br = new BufferedReader(new
    InputStreamReader(System.in),1);
    String string="";
    try {
        string = br.readLine(); }
    catch (IOException ex) {
        System.out.println(ex); }
    return string; }

/**
```

```
 * Metodo que lee un dato tipo int desde el teclado.
 * @return Integer devuelve el entero leído a traves de la pantalla
 * @throws NumberFormatException arroja esta excepción cuando el formato del
número no es correcto
 * Postcondition: El valor que devuelve debe ser un número entero
 */
public static int readInt() throws NumberFormatException{
    return Integer.parseInt(readString());
}

/**
 * Metodo que lee un fichero que este en la carpeta raíz del proyecto.
 * @param nombreFichero es el string que contiene el nombre del fichero
 * @return v ArrayList de Strings que contiene la información del fichero
leído
 */
public static ArrayList <String> leeFichero(String nombreFichero){
    ArrayList <String> v = new ArrayList <String>();
    File fichero=null;
    FileReader fr=null;
    BufferedReader br=null;
    try{
        fichero=new File(nombreFichero);
        fr=new FileReader(fichero);
        br=new BufferedReader(fr);
        String linea;
        while ((linea=br.readLine())!=null){
        v.add(linea);}
    }
    catch (Exception e){
        e.printStackTrace();
    }
    finally {
    try {
        if (null!= fr){
        fr.close();
        br.close();}
        }
        catch (Exception e1){
        e1.printStackTrace();
        }
    }
    return v;
    }
/**
 * Metodo que realiza la serialización y deserialización.
 * @param <A> parametro descocido para mi
 * @param a parametro descocido para mi
```

```
 * @param nombreFichero nombre del fichero a guardar en en la carpeta raíz
del proyecto desde el programa o cargar desde la carpeta raíz hacía el
programa
 */
    public static <A> void serialize(A a, String nombreFichero) {
        System.out.println("Serializando...");
        try {
            FileOutputStream fos = new FileOutputStream(nombreFichero) ;
            ObjectOutputStream oos = new ObjectOutputStream(fos) ;
            oos.writeObject(a) ;
        } catch (Exception e) {
            System.err.println("Problem: "+e) ;
        }
    }
    public static <A> A deserialize(String nombreFichero) {
        System.out.println("DeSerializing...");
        try {
            FileInputStream fis = new FileInputStream(nombreFichero) ;
            ObjectInputStream iis = new ObjectInputStream(fis) ;
            return (A) iis.readObject() ;
        } catch (Exception e) {
            System.err.println("Problem: "+e) ;
        }
        return null ;
}

}
```

Clase MenuPrincipal:

```
package Menu;

import EntradaSalida.MyInput;
import java.util.logging.Level;
import java.util.logging.Logger;
import librerias.estructurasDeDatos.jerarquicos.*;
import librerias.excepciones.ElementoDuplicado;

/**
 * Esta clase contiene los metodos y atributos del menú principal
 * @author Raúl Colindres y Alfredo Sobrados
 */
public class MenuPrincipal extends Menus{
    private ABBEnteros arbolEnteros;
    /**
     * Método constructor sin parametros
     */
    public MenuPrincipal(){
        super();
        arbolEnteros = new ABBEnteros();
    }
```

```
    /**
     * Método que ejecuta el menú principal del programa.
     * Realiza un bucle donde se ejecutan las opciones seleccionadas por el
usuario hasta que la respuesta sea diferente de "s".
     */
    @Override
    public void ejecutar(){
        String respuesta="";

        do{
            respuesta = ejecutarOpciones();
        }while(respuesta.equals("s"));
    }
    /**
     * Método que ejecuta una opción del menú principal.
     * Imprime el menú principal y solicita al usuario que seleccione una
opción.
     * Si la opción ingresada es inválida, muestra un mensaje de error.
     * @return La respuesta del usuario ("s" para seguir en el menú, "n"
para salir).
     */
    @Override
    public String ejecutarOpciones(){
        System.out.println("");
            System.out.println("Menú Principal");
            System.out.println("seleccione una opción:");
            System.out.println("0. Salir");
            System.out.println("1. Crear ABB de enteros positivos
(Equilibrado)");
            System.out.println("2. Listado de claves en orden ascendente");
            System.out.println("3. Listado de claves en orden descendente");
            System.out.println("4. Listado de claves por niveles
(converso)");
            System.out.println("5. Transformar árbol ABB de enteros");
            System.out.println("6. Sumar claves menores que un número
dado");
            System.out.println("7. Calcular el antecesor de un nodo");
            System.out.println("8. Mostrar hojas (posiciones impares)");

        String s=MyInput.readString();

        int i=0;
        try{
            i= Integer.parseInt(s);
        }catch(NumberFormatException ex){
            System.out.println("La entrada no tiene formato de número.
Inténtelo de nuevo");
            return "s";
        }
        switch(i){
```

```
            case 0: {System.out.println("Gracias por utilizar nuestros TAD
ABB y ABBEnteros");return "n";}
            case 1: {try {
                crearABBEnterosPos();
            } catch (ElementoDuplicado ex) {

Logger.getLogger(MenuPrincipal.class.getName()).log(Level.SEVERE, null, ex);
            }
            return "s";}
            case 2: {listadoClavesOrdenAscendente();return "s";}
            case 3: {listadoClavesOrdenDescendente();return "s";}
            case 4: {listadoClavesPorNivelesInverso();return "s";}
            case 5: {transformarABBEnteros();return "s";}
            case 6: {sumarClavesMinNumDado();return "s";}
            case 7: {calcularAntecesorNodo();return "s";}
            case 8: {mostrarHojasPosImpares();return "s";}
            default: {System.out.println("Opción no válida. Inténtelo de
nuevo."); return "s";}
        }
    }
    /**
     * Método que crea el arból binario de busqueda de enteros
     * @throws ElementoDuplicado si se intenta insertar un elemento
duplicado
     */
    public void crearABBEnterosPos() throws ElementoDuplicado{
        Integer v[] = new Integer[50];

        String respuesta;
        int clave, contador=0;

        System.out.println("Crear un ABB(Árbol Binario de Busqueda) de
claves enteras equilibrado (MÁX:50)");
        do{
            System.out.println("Introduzca la clave entera que quiere añadir
al árbol");
            clave = MyInput.readInt();
            v[contador] = clave;
            contador++;

            System.out.println("¿Quieres añadir mas claves al árbol?(S/N):
");
            respuesta = MyInput.readString().toUpperCase();
        }while(respuesta.equals("S") && contador < 50);

        if(contador==50){
            System.out.println("El array ha llegado a su máxima capacidad no
se pueden añadir mas claves enteras en el árbol binario de busqueda");
        }
        arbolEnteros.insertarEquilibrado(v);
```

```
    }
    /**
     * Método que lista las claves del ABB de enteros de manera ascendente
     */
    public void listadoClavesOrdenAscendente(){
        System.out.println("Opción 2: Listado de claves en orden
ascendente");
        System.out.println("Salida en pantalla:");
        System.out.print(arbolEnteros.toStringInOrden());
    }
    /**
     * Método que lista las claves del ABB de enteros de manera descendente
     */
    public void listadoClavesOrdenDescendente(){
        System.out.println("Opción 3: Listado de claves en orden
descendente");
        System.out.println("Salida en pantalla:");
        System.out.print(arbolEnteros.toStringInOrdenInverso());
    }
    /**
     * Método que lista las claves del ABB de enteros por niveles inverso
     */
    public void listadoClavesPorNivelesInverso(){
        System.out.println("Opción 4: Listado de claves por niveles
(converso)");
        System.out.println("Salida en pantalla:");
        System.out.print(arbolEnteros.toStringPorNivelesInverso());
    }
    /**
     * Método que transforma las claves del ABB de enteros de claves
positivas a negativas
     */
    public void transformarABBEnteros(){
        System.out.println("Opción 5: Transformar árbol ABB de enteros");
        arbolEnteros.transformarABBEnteros();
        System.out.println("Árbol transformado!!");
    }
    /**
     * Método que suma las claves del ABB de enteros a partir de un número
mínimo dado
     */
    public void sumarClavesMinNumDado(){
        System.out.println("Opción 6: Sumar claves menores que un número
dado");
        arbolEnteros.sumarClavesMinNumDado();
    }
    /**
     * Método que calcula el antecesor de una clave del ABB de enteros
     */
    public void calcularAntecesorNodo(){
```

```
        System.out.println("Opción 7: Calcular el antecesor de un nodo");
        arbolEnteros.calcularAntecesorNodo();
    }
    /**
     * Método que muestra las hojas en las posiciones impares
     */
    public void mostrarHojasPosImpares(){
        System.out.println("Opción 8: Mostrar hojas (posiciones impares)");
        arbolEnteros.mostrarHojasPosImpares();
    }
}
```

Clase Menus:

```
package Menu;

/**
 * Esta clase contiene los metodos y atributos de los menus
 * @author Raúl Colindres y Alfredo Sobrados
 */
public abstract class Menus {
    /**
     * Método que ejecuta el menú principal del programa.
     * Realiza un bucle donde se ejecutan las opciones seleccionadas por el
usuario hasta que la respuesta sea diferente de "s".
     */
    public void ejecutar(){
        String respuesta="";

        do{
            respuesta = ejecutarOpciones();
        }while(respuesta.equals("s"));
    }
    /**
     * Ejecuta las opciones disponibles y devuelve un resultado en forma de
cadena.
     * La implementación de este método depende de las funcionalidades
específicas
     * de la clase MenuPrincipal que es la que lo implementa.
     * @return el resultado de la ejecución de las opciones en forma de
cadena
     */
    public abstract String ejecutarOpciones();
}
```

Clase ABB:

```
package librerias.estructurasDeDatos.jerarquicos;

import java.lang.reflect.Array;
import librerias.estructurasDeDatos.lineales.ArrayCola;
import librerias.estructurasDeDatos.modelos.*;
import librerias.excepciones.*;
import java.util.Arrays;
import java.util.Objects;

/**
 * Esta clase contiene los metodos y atributos del árbol binario de busqueda
 * @author Raúl Colindres y Alfredo Sobrados
 * @param <E> tipo génerico que hereda de la interfaz Comparable
 */
public class ABB<E extends Comparable<E>> implements Modelo_ABB<E>{
    protected NodoABB<E> raiz;
    /**
     * Método constructor
     */
    public ABB(){
        raiz=null;
    }
    /**
     * Método constructor parametrizado
     * @param x la raíz del arbol binario de busqueda de enteros
     */
    public ABB(E x){
        raiz = new NodoABB<E>(x);
    }
    /**
     * Método lanzadera que recupera un elemento del ABB
     * @param x elemento a recuperar del ABB
     * @return devuelve el elemento recuperado del ABB (si lo recupera)
     * @throws ElementoNoEncontrado si se intenta recuperar un elemento que
no esta en el árbol binario de busqueda
     */
    @Override
    public E recuperar(E x) throws ElementoNoEncontrado {
        NodoABB<E> res = recuperar(x, this.raiz);
        if(res == null)
            throw new ElementoNoEncontrado("El dato "+x+" no está");
        return res.dato;
    }
    /**
     * Método privado recursivo que recupera un elemento del ABB
     * @param x elemento a recuperar del ABB
     * @param n raíz del ABB
     * @return devuelve el NodoABB que contiene el elemento a recuperar
```

```
     */
    private NodoABB<E> recuperar(E x, NodoABB<E> n){
        NodoABB<E> res = n;
        if(n != null){
            int resC = n.dato.compareTo(x);
            if( resC < 0) res = recuperar(x, n.der);
            else if(resC > 0) res = recuperar(x, n.izq);
        }
        return res;
    }
    /**
     * Método que recupera el valor mínimo almacenado en el árbol binario de
busqueda
     * @return el valor mínimo del árbol binario de busqueda
     */
    @Override
    public E recuperarMin() {
        return recuperarMin(this.raiz).dato;
    }
    /**
     * Método que recupera el nodo que contiene el valor mínimo a partir del
nodo actual.
     * @param actual el nodo actual desde donde comenzar la búsqueda
     * @return el nodo que contiene el valor mínimo
     */
    private NodoABB<E> recuperarMin(NodoABB<E> actual){
        if(actual.izq == null)
            return actual;
        else
            return(recuperarMin(actual.izq));
    }
    /**
     * Método que recupera el valor máximo almacenado en el árbol binario de
busqueda.
     * @return el valor máximo del árbol binario de busqueda
     */
    @Override
    public E recuperarMax(){
        return recuperarMax(this.raiz).dato;
    }
    /**
     * Método que recupera el nodo que contiene el valor máximo a partir del
nodo actual.
     * @param actual el nodo actual desde donde comenzar la búsqueda
     * @return el nodo que contiene el valor máximo
     */
    public NodoABB<E> recuperarMax(NodoABB<E> actual){
        return(actual.der == null)?actual:recuperarMax(actual.der);
    }
    /**
```

```
     * Método que inserta un elemento en el árbol binario de busqueda sin
permitir duplicados.
     * @param x el elemento a insertar
     * @throws ElementoDuplicado si se intenta insertar un elemento
duplicado
     */
    @Override
    public void insertarSinDuplicados(E x) throws ElementoDuplicado {
        if (x != null) { // Ignorar elementos nulos
            this.raiz = insertarSinDuplicados(x, raiz);
        }
    }
    /**
     * Método que inserta un elemento en el árbol binario de busqueda sin
permitir duplicados, comenzando desde el nodo actual.
     * @param x el elemento a insertar
     * @param actual el nodo actual desde donde comenzar la inserción
     * @return el nodo actualizado después de la inserción
     * @throws ElementoDuplicado si se intenta insertar un elemento
duplicado
     */
    private NodoABB<E> insertarSinDuplicados(E x, NodoABB<E> actual) throws
ElementoDuplicado {
        NodoABB<E> res = actual;
        if(actual == null){
            res = new NodoABB<E>(x);
        }else{
            if(actual.dato != null){
                int resC = actual.dato.compareTo(x);
                if(resC == 0) throw new ElementoDuplicado(x+" está
duplicado");
                if(resC < 0) res.der = insertarSinDuplicados(x, actual.der);
                else res.izq = insertarSinDuplicados(x, actual.izq);
                actual.tamanyo++;
            }
        }
        return res;
    }
    /**
     * Método que inserta un elemento en el árbol binario de busqueda
permitiendo duplicados.
     * @param x el elemento a insertar
     */
    @Override
    public void insertarConDuplicados(E x) {
        this.raiz = insertarConDuplicados(x, raiz);
    }
    /**
     * Método que inserta un elemento en el árbol binario de busqueda
permitiendo duplicados, comenzando desde el nodo actual.
```

```
     * @param x el elemento a insertar
     * @param actual el nodo actual desde donde comenzar la inserción
     * @return el nodo actualizado después de la inserción
     */
    private NodoABB<E> insertarConDuplicados(E x, NodoABB<E> actual){
        NodoABB<E> res = actual;
        if(actual == null){
            res = new NodoABB<E>(x);
        }else{
            int resC = actual.dato.compareTo(x);

            actual.tamanyo++;
            if(resC < 0) res.der=insertarConDuplicados(x, actual.der);
            else res.izq = insertarConDuplicados(x, actual.izq);
        }
        return res;
    }
    /**
     * Método que inserta un array de elementos en el árbol binario de
busqueda de forma equilibrada.
     * @param v el array de elementos a insertar
     * @throws ElementoDuplicado si se intenta insertar un elemento
duplicado
     */
    public void insertarEquilibrado(E v[]) throws ElementoDuplicado{
        E[] resultado = Arrays.stream(v)
            .filter(Objects::nonNull)
            .toArray(size -> (E[])
Array.newInstance(v.getClass().getComponentType(), size));

        Arrays.sort(resultado);
        insertarEquilibrado(resultado, 0, resultado.length - 1);
    }
    /**
     * Método que inserta un array de elementos en el árbol binario de
busqueda de forma equilibrada, dentro del rango especificado.
     * @param v el array de elementos a insertar
     * @param izq el índice izquierdo del rango
     * @param der el índice derecho del rango
     * @throws ElementoDuplicado si se intenta insertar un elemento
duplicado
     */
    private void insertarEquilibrado(E v[], int izq, int der) throws
ElementoDuplicado{
        if(izq <= der){
            int med = (izq + der) / 2;
            raiz = insertarSinDuplicados(v[med], raiz);
            insertarEquilibrado(v, izq, med - 1);
            insertarEquilibrado(v, med + 1, der);
        }
```

```
    }
    /**
     * Método que elimina un elemento del árbol binario de busqueda.
     * @param x el elemento a eliminar
     * @throws ElementoNoEncontrado si el elemento no se encuentra en el
árbol binario de busqueda de enteros
     */
    @Override
    public void eliminar(E x) throws ElementoNoEncontrado {
        this.raiz = eliminar(x, this.raiz);
    }
    /**
     * Método que elimina un elemento del árbol binario de busqueda
comenzando desde el nodo actual.
     * @param x el elemento a eliminar
     * @param actual el nodo actual desde donde comenzar la eliminación
     * @return el nodo actualizado después de la eliminación
     * @throws ElementoNoEncontrado si el elemento no se encuentra en el
árbol binario de busqueda de enteros
     */
    private NodoABB<E> eliminar(E x, NodoABB<E> actual) throws
ElementoNoEncontrado {
        NodoABB<E> res = actual;
        if(actual == null) throw new ElementoNoEncontrado("El dato "+x+" no
está");
        int resC = actual.dato.compareTo(x);
        if(resC < 0) res.der = eliminar(x, actual.der);
        else if (resC > 0) res.izq = eliminar(x, actual.izq);
        else{
            if(actual.izq != null && actual.der != null){
                res.dato = recuperarMin(actual.der).dato;
                res.der = eliminarMin(actual.der);
            }
            else res = (actual.izq != null)?actual.izq:actual.der;
        }
        actual.tamanyo--;
        return res;
    }
    /**
     * Método que elimina el elemento mínimo del árbol binario de busqueda y
lo devuelve.
     * @return el elemento mínimo del árbol binario de busqueda de enteros
     */
    @Override
    public E eliminarMin() {
        E min = recuperarMin();
        this.raiz = eliminarMin(this.raiz);
        return min;
    }
    /**
```

```
     * Método que elimina el elemento mínimo del árbol binario de busqueda
comenzando desde el nodo actual.
     * @param actual el nodo actual desde donde comenzar la eliminación
     * @return el nodo actualizado después de la eliminación
     */
    private NodoABB<E> eliminarMin(NodoABB<E> actual){
        if(actual.izq != null){
            actual.tamanyo--; actual.izq = eliminarMin(actual.izq);
        }else{
            actual = actual.der;
        }
        return actual;
    }
     /**
     * Método que devuelve el tamaño (cantidad de elementos) del árbol
binario de busqueda.
     * @return el tamaño del árbol binario de busqueda
     */
    @Override
    public int tamanyo() {
        return tamanyo(this.raiz);
    }
    /**
     * Método que calcula el tamaño (cantidad de elementos) del árbol
binario de busqueda comenzando desde el nodo actual.
     * @param actual el nodo actual desde donde comenzar el cálculo
     * @return el tamaño del árbol binario de busqueda
     */
    private int tamanyo(NodoABB<E> actual){
        return(actual!=null)?actual.tamanyo:0;
    }
    /**
     * Método que devuelve la altura del árbol binario de busqueda.
     * @return la altura del árbol binario de busqueda
     */
    @Override
    public int altura() {
        return altura(this.raiz);
    }
    /**
     * Método que calcula la altura del árbol binario de busqueda comenzando
desde el nodo actual.
     * @param actual el nodo actual desde donde comenzar el cálculo
     * @return la altura del árbol binario de busqueda
     */
    private int altura(NodoABB<E> actual){
        if(actual == null)
            return -1;
        else
            return (1 + Math.max(altura(actual.izq), altura(actual.der)));
```

```
    }
    /**
     * Método que verifica si el árbol binario de busqueda está vacío.
     * @return true si el árbol binario de busqueda está vacío, false en
caso contrario
     */
    @Override
    public boolean esVacio() {
        return raiz == null;
    }
    /**
     * Método que devuelve un String que representa el árbol binario de
busqueda en recorrido PreOrden.
     * @return representación del árbol binario de busqueda en recorrido
PreOrden
     */
    @Override
    public String toStringPreOrden() {
        if(this.raiz != null) return toStringPreOrden(this.raiz); else
return "*";
    }
    /**
     * Método que devuelve un String que representa el árbol binario de
busqueda en recorrido PreOrden, comenzando desde el nodo actual.
     * @param actual el nodo actual desde donde comenzar el recorrido
PreOrden
     * @return representación del árbol binario de busqueda en recorrido
PreOrden
     */
    private String toStringPreOrden(NodoABB<E> actual){
        String res="";
        if(actual != null){
            if(actual.dato != null){
                res += actual.dato.toString()+"\n";
            }
            if(actual.izq != null) res += toStringPreOrden(actual.izq);
            if(actual.der != null) res += toStringPreOrden(actual.der);
        }

        return res;
    }
    /**
     * Método que devuelve un String que representa el árbol binario de
busqueda en recorrido PostOrden.
     * @return representación del árbol binario de busqueda en recorrido
PostOrden
     */
    @Override
    public String toStringPostOrden() {
        if(this.raiz != null) return toStringPostOrden(this.raiz);
```

```
        else return "*";
    }
    /**
     * Método que devuelve un String que representa el árbol binario de
busqueda en recorrido PostOrden, comenzando desde el nodo actual.
     * @param actual el nodo actual desde donde comenzar el recorrido
PostOrden
     * @return representación del árbol binario de busqueda en recorrido
PostOrden
     */
    private String toStringPostOrden(NodoABB<E> actual){
        String res = "";
        if(actual.izq != null) res += toStringPostOrden(actual.izq);
        if(actual.der != null) res += toStringPostOrden(actual.der);
        res += actual.dato.toString()+"\n";
        return res;
    }
    /**
     * Método que devuelve un String que representa el árbol binario de
busqueda en recorrido InOrden.
     * @return representación del árbol en recorrido InOrden
     */
    @Override
    public String toStringInOrden() {
        if(this.raiz != null) return toStringInOrden(this.raiz); else return
"*";
    }
    /**
     * Método que devuelve un String que representa el árbol binario de
busqueda en recorrido InOrden, comenzando desde el nodo actual.
     * @param actual el nodo actual desde donde comenzar el recorrido
InOrden
     * @return representación del árbol binario de busqueda en recorrido
InOrden
     */
    private String toStringInOrden(NodoABB<E> actual){
        String res = "";
        if(actual.izq != null) res+= toStringInOrden(actual.izq);
        res+= actual.dato.toString()+"\n";
        if(actual.der != null) res += toStringInOrden(actual.der);
        return res;
    }
    /**
     * Método que devuelve un String que representa el árbol binario de
busqueda en recorrido por niveles.
     * @return representación del árbol binario de busqueda en recorrido por
niveles
     */
    @Override
    public String toStringPorNiveles() {
```

```
        if(this.raiz != null) return toStringPorNiveles(this.raiz);
        else return "*";
    }
    /**
     * Método que devuelve un String que representa el árbol binario de
busqueda en recorrido por niveles, comenzando desde el nodo actual.
     * @param actual el nodo actual desde donde comenzar el recorrido por
niveles
     * @return representación del árbol binario de busqueda en recorrido por
niveles
     */
    private String toStringPorNiveles(NodoABB<E> actual){
        Cola<NodoABB<E>> q = new ArrayCola<NodoABB<E>>();
        q.encolar(actual); String res = "";
        while(!q.esVacia()){
            NodoABB<E> nodoActual = q.desencolar();
            res += nodoActual.dato.toString()+"\n";
            if(nodoActual.izq != null) q.encolar(nodoActual.izq);
            if(nodoActual.der != null) q.encolar(nodoActual.der);
        }
        return res;
    }
}
```

Clase ABBEnteros:

```
package librerias.estructurasDeDatos.jerarquicos;

import EntradaSalida.MyInput;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.logging.Level;
import java.util.logging.Logger;
import librerias.estructurasDeDatos.lineales.ArrayCola;
import librerias.estructurasDeDatos.modelos.*;
import librerias.excepciones.*;

/**
 * Esta clase contiene los metodos y atributos del árbol binario de busqueda
de enteros
 * @author Raúl Colindres y Alfredo Sobrados
 */
public class ABBEnteros implements Modelo_ABBEnteros<Integer>{
    protected NodoABB raiz;
    /**
     * Método constructor
     */
    public ABBEnteros(){
        raiz=null;
    }
```

```
    /**
     * Método constructor parametrizado
     * @param x la raíz del arbol binario de busqueda de enteros
     */
    public ABBEnteros(Integer x){
        raiz = new NodoABB(x);
    }
    /**
     * Método lanzadera que recupera un entero del ABB
     * @param x entero a recuperar del ABB
     * @return devuelve el entero recuperado del ABB (si lo recupera)
     * @throws ElementoNoEncontrado si se intenta recuperar un elemento que
no esta en el árbol binario de busqueda de enteros
     */
    @Override
    public Integer recuperar(Integer x) throws ElementoNoEncontrado {
        NodoABB res = recuperar(x, this.raiz);
        if(res == null)
            throw new ElementoNoEncontrado("El dato "+x+" no está");
        return (Integer) res.dato;
    }
    /**
     * Método privado recursivo que recupera un entero del ABB
     * @param x entero a recuperar del ABB
     * @param n raíz del ABB
     * @return devuelve el NodoABB que contiene el entero a recuperar
     */
    private NodoABB recuperar(Integer x, NodoABB<Integer> n){
        NodoABB res = n;
        if(n != null){
            int resC = n.dato.compareTo(x);
            if( resC < 0) res = recuperar(x, n.der);
            else if(resC > 0) res = recuperar(x, n.izq);
        }
        return res;
    }
    /**
     * Método que recupera el valor mínimo almacenado en el árbol binario de
busqueda de enteros
     * @return el valor mínimo del árbol binario de busqueda de enteros
     */
    @Override
    public Integer recuperarMin() {
        return (Integer) recuperarMin(this.raiz).dato;
    }
    /**
     * Método que recupera el nodo que contiene el valor mínimo a partir del
nodo actual.
     * @param actual el nodo actual desde donde comenzar la búsqueda
     * @return el nodo que contiene el valor mínimo
```

```
     */
    private NodoABB<Integer> recuperarMin(NodoABB<Integer> actual){
        if(actual.izq == null)
            return actual;
        else
            return(recuperarMin(actual.izq));
    }
    /**
     * Método que recupera el valor máximo almacenado en el árbol binario de
busqueda de enteros.
     * @return el valor máximo del árbol binario de busqueda de enteros
     */
    @Override
    public Integer recuperarMax(){
        return (Integer) recuperarMax(this.raiz).dato;
    }
    /**
     * Método que recupera el nodo que contiene el valor máximo a partir del
nodo actual.
     * @param actual el nodo actual desde donde comenzar la búsqueda
     * @return el nodo que contiene el valor máximo
     */
    public NodoABB recuperarMax(NodoABB<Integer> actual){
        return(actual.der == null)?actual:recuperarMax(actual.der);
    }
    /**
     * Método que inserta un elemento en el árbol binario de busqueda de
enteros sin permitir duplicados.
     * @param x el entero a insertar
     * @throws ElementoDuplicado si se intenta insertar un elemento
duplicado
     */
    @Override
    public void insertarSinDuplicados(Integer x) throws ElementoDuplicado {
        if (x != null) { // Ignorar elementos nulos
            this.raiz = insertarSinDuplicados(x, raiz);
        }
    }
    /**
     * Método que inserta un elemento en el árbol binario de busqueda de
enteros sin permitir duplicados, comenzando desde el nodo actual.
     * @param x el entero a insertar
     * @param actual el nodo actual desde donde comenzar la inserción
     * @return el nodo actualizado después de la inserción
     * @throws ElementoDuplicado si se intenta insertar un elemento
duplicado
     */
    private NodoABB insertarSinDuplicados(Integer x, NodoABB<Integer>
actual) throws ElementoDuplicado {
        NodoABB res = actual;
```

```
        if(actual == null){
            res = new NodoABB(x);
        }else{
            if(actual.dato != null){
                int resC = actual.dato.compareTo(x);
                if(resC == 0) throw new ElementoDuplicado(x+" está
duplicado");
                if(resC < 0) res.der = insertarSinDuplicados(x, actual.der);
                else res.izq = insertarSinDuplicados(x, actual.izq);
                actual.tamanyo++;
            }
        }
        return res;
    }
    /**
     * Método que inserta un elemento en el árbol binario de busqueda de
enteros permitiendo duplicados.
     * @param x el elemento a insertar
     */
    @Override
    public void insertarConDuplicados(Integer x) {
        this.raiz = insertarConDuplicados(x, raiz);
    }
    /**
     * Método que inserta un elemento en el árbol binario de busqueda de
enteros permitiendo duplicados, comenzando desde el nodo actual.
     * @param x el elemento a insertar
     * @param actual el nodo actual desde donde comenzar la inserción
     * @return el nodo actualizado después de la inserción
     */
    private NodoABB insertarConDuplicados(Integer x, NodoABB actual){
        NodoABB res = actual;
        if(actual == null){
            res = new NodoABB(x);
        }else{
            int resC = (int) actual.dato -  x;

        actual.tamanyo++;
        if (resC < 0) {
            res.der = insertarConDuplicados(x, actual.der);
        } else {
            res.izq = insertarConDuplicados(x, actual.izq);
        }
        }
        return res;
    }
    /**
     * Método que inserta un array de elementos en el árbol binario de
busqueda de enteros de forma equilibrada.
     * @param v el array de elementos a insertar
```

```
     * @throws ElementoDuplicado si se intenta insertar un elemento
duplicado
     */
    public void insertarEquilibrado(Integer v[]) throws ElementoDuplicado{
        ArrayList<Integer> lista = new ArrayList<>();

        for (Integer v1 : v) {
            if (v1 != null) {
                lista.add(v1);
            }
        }

        int a[] = new int[lista.size()];

        for(int i = 0; i< lista.size(); i++){
            a[i] = lista.get(i);
        }

        Arrays.sort(a);

        Integer[] r = new Integer[a.length];

        for(int i = 0; i < a.length; i++){
            r[i] = a[i];
        }

        insertarEquilibrado(r, 0, r.length - 1);
    }
    /**
     * Método que inserta un array de elementos en el árbol binario de
busqueda de enteros de forma equilibrada, dentro del rango especificado.
     * @param v el array de elementos a insertar
     * @param izq el índice izquierdo del rango
     * @param der el índice derecho del rango
     * @throws ElementoDuplicado si se intenta insertar un elemento
duplicado
     */
    private void insertarEquilibrado(Integer v[], int izq, int der) throws
ElementoDuplicado{
        if(izq <= der){
            int med = (izq + der) / 2;
            raiz = insertarSinDuplicados(v[med], raiz);
            insertarEquilibrado(v, izq, med - 1);
            insertarEquilibrado(v, med + 1, der);
        }
    }
    /**
     * Método que elimina un elemento del árbol binario de busqueda de
enteros.
     * @param x el elemento a eliminar
```

```
     * @throws ElementoNoEncontrado si el elemento no se encuentra en el
árbol binario de busqueda de enteros
     */
    @Override
    public void eliminar(Integer x) throws ElementoNoEncontrado {
        this.raiz = eliminar(x, this.raiz);
    }
    /**
     * Método que elimina un elemento del árbol binario de busqueda de
enteros comenzando desde el nodo actual.
     * @param x el elemento a eliminar
     * @param actual el nodo actual desde donde comenzar la eliminación
     * @return el nodo actualizado después de la eliminación
     * @throws ElementoNoEncontrado si el elemento no se encuentra en el
árbol binario de busqueda de enteros
     */
    public NodoABB eliminar(Integer x, NodoABB<Integer> actual) throws
ElementoNoEncontrado {
        NodoABB res = actual;
        if(actual == null) throw new ElementoNoEncontrado("El dato "+x+" no
está");
        int resC = actual.dato.compareTo(x);
        if(resC < 0) res.der = eliminar(x, actual.der);
        else if (resC > 0) res.izq = eliminar(x, actual.izq);
        else{
            if(actual.izq != null && actual.der != null){
                res.dato = recuperarMin(actual.der).dato;
                res.der = eliminarMin(actual.der);
            }
            else res = (actual.izq != null)?actual.izq:actual.der;
        }
        actual.tamanyo--;
        return res;
    }
    /**
     * Método que elimina el elemento mínimo del árbol binario de busqueda
de enteros y lo devuelve.
     * @return el elemento mínimo del árbol binario de busqueda de enteros
     */
    @Override
    public Integer eliminarMin() {
        Integer min = recuperarMin();
        this.raiz = eliminarMin(this.raiz);
        return min;
    }
    /**
     * Método que elimina el elemento mínimo del árbol binario de busqueda
de enteros comenzando desde el nodo actual.
     * @param actual el nodo actual desde donde comenzar la eliminación
     * @return el nodo actualizado después de la eliminación
```

```
     */
    private NodoABB eliminarMin(NodoABB actual){
        if(actual.izq != null){
            actual.tamanyo--; actual.izq = eliminarMin(actual.izq);
        }else{
            actual = actual.der;
        }
        return actual;
    }
    /**
     * Método que devuelve el tamaño (cantidad de elementos) del árbol
binario de busqueda de enteros.
     * @return el tamaño del árbol binario de busqueda de enteros
     */
    @Override
    public int tamanyo() {
        return tamanyo(this.raiz);
    }
    /**
     * Método que calcula el tamaño (cantidad de elementos) del árbol
binario de busqueda de enteros comenzando desde el nodo actual.
     * @param actual el nodo actual desde donde comenzar el cálculo
     * @return el tamaño del árbol binario de busqueda de enteros
     */
    private int tamanyo(NodoABB actual){
        return(actual!=null)?actual.tamanyo:0;
    }
    /**
     * Método que devuelve la altura del árbol binario de busqueda de
enteros.
     * @return la altura del árbol binario de busqueda de enteros
     */
    @Override
    public int altura() {
        return altura(this.raiz);
    }
    /**
     * Método que calcula la altura del árbol binario de busqueda de enteros
comenzando desde el nodo actual.
     * @param actual el nodo actual desde donde comenzar el cálculo
     * @return la altura del árbol binario de busqueda de enteros
     */
    private int altura(NodoABB actual){
        if(actual == null)
            return -1;
        else
            return (1 + Math.max(altura(actual.izq), altura(actual.der)));
    }
    /**
```

```
     * Método que verifica si el árbol binario de busqueda de enteros está
vacío.
     * @return true si el árbol binario de busqueda de enteros está vacío,
false en caso contrario
     */
    @Override
    public boolean esVacio() {
        return raiz == null;
    }
    /**
     * Método que devuelve un String que representa el árbol binario de
busqueda de enteros en recorrido PreOrden.
     * @return representación del árbol binario de busqueda de enteros en
recorrido PreOrden
     */
    @Override
    public String toStringPreOrden() {
        if(this.raiz != null) return toStringPreOrden(this.raiz); else
return "*";
    }
    /**
     * Método que devuelve un String que representa el árbol binario de
busqueda de enteros en recorrido PreOrden, comenzando desde el nodo actual.
     * @param actual el nodo actual desde donde comenzar el recorrido
PreOrden
     * @return representación del árbol binario de busqueda de enteros en
recorrido PreOrden
     */
    private String toStringPreOrden(NodoABB actual){
        String res="";
        if(actual != null){
            res = actual.dato.toString()+"\n";
        }
        if(actual.izq != null) res += toStringPreOrden(actual.izq);
        if(actual.der != null) res += toStringPreOrden(actual.der);
        return res;
    }
    /**
     * Método que devuelve un String que representa el árbol binario de
busqueda de enteros en recorrido PostOrden.
     * @return representación del árbol binario de busqueda de enteros en
recorrido PostOrden
     */
    @Override
    public String toStringPostOrden() {
        if(this.raiz != null) return toStringPostOrden(this.raiz);
        else return "*";
    }
    /**
```

```
     * Método que devuelve un String que representa el árbol binario de
busqueda de enteros en recorrido PostOrden, comenzando desde el nodo actual.
     * @param actual el nodo actual desde donde comenzar el recorrido
PostOrden
     * @return representación del árbol binario de busqueda de enteros en
recorrido PostOrden
     */
    private String toStringPostOrden(NodoABB actual){
        String res = "";
        if(actual.izq != null) res += toStringPostOrden(actual.izq);
        if(actual.der != null) res += toStringPostOrden(actual.der);
        if(actual != null){
           res+= actual.dato.toString()+"\n";
        }
        return res;
    }
    /**
     * Método que devuelve un String que representa el árbol binario de
busqueda de enteros en recorrido InOrden.
     * @return representación del árbol en recorrido InOrden
     */
    @Override
    public String toStringInOrden() {
        if(this.raiz != null) return toStringInOrden(this.raiz); else return
"*";
    }
    /**
     * Método que devuelve un String que representa el árbol binario de
busqueda de enteros en recorrido InOrden, comenzando desde el nodo actual.
     * @param actual el nodo actual desde donde comenzar el recorrido
InOrden
     * @return representación del árbol binario de busqueda de enteros en
recorrido InOrden
     */
    private String toStringInOrden(NodoABB actual){
        String res = "";
        if(actual.izq != null) res+= toStringInOrden(actual.izq);
        if(actual != null){
            res+= actual.dato.toString()+"\n";
        }
        if(actual.der != null) res += toStringInOrden(actual.der);
        return res;
    }
    /**
     * Método que devuelve un String que representa el árbol binario de
busqueda de enteros en recorrido InOrden inverso.
     * @return representación del árbol binario de busqueda de enteros en
recorrido InOrden inverso
     */
    @Override
```

```
    public String toStringInOrdenInverso() {
        if(this.raiz != null) return toStringInOrdenInverso(this.raiz); else
return "*";
    }
    /**
     * Método que devuelve un String que representa el árbol binario de
busqueda de enteros en recorrido InOrden inverso, comenzando desde el nodo
actual.
     * @param actual el nodo actual desde donde comenzar el recorrido
InOrden inverso
     * @return representación del árbol binario de busqueda de enteros en
recorrido InOrden inverso
     */
    private String toStringInOrdenInverso(NodoABB actual){
        String res="";
        if(actual != null){
            if(actual.der != null) res+= toStringInOrdenInverso(actual.der);
            if(actual.dato != null){
                res+= actual.dato.toString()+"\n";
            }
            if(actual.izq != null) res +=
toStringInOrdenInverso(actual.izq);
        }
        return res;
    }
    /**
     * Método que devuelve un String que representa el árbol binario de
busqueda de enteros en recorrido por niveles.
     * @return representación del árbol binario de busqueda de enteros en
recorrido por niveles
     */
    @Override
    public String toStringPorNiveles() {
         if(this.raiz != null) return toStringPorNiveles(this.raiz);
         else return "*";
    }
    /**
     * Método que devuelve un String que representa el árbol binario de
busqueda de enteros en recorrido por niveles, comenzando desde el nodo
actual.
     * @param actual el nodo actual desde donde comenzar el recorrido por
niveles
     * @return representación del árbol binario de busqueda de enteros en
recorrido por niveles
     */
    private String toStringPorNiveles(NodoABB actual){
        ArrayCola<NodoABB> q = new ArrayCola<NodoABB>();
        q.encolar(actual); String res = "";
        while(!q.esVacia()){
            NodoABB nodoActual = q.desencolar();
```

```
            res += nodoActual.dato.toString()+"\n";
            if(nodoActual.izq != null) q.encolar(nodoActual.izq);
            if(nodoActual.der != null) q.encolar(nodoActual.der);
        }
        return res;
    }
    /**
     * Método que devuelve un String que representa el árbol binario de
busqueda de enteros en recorrido por niveles inverso.
     * @return representación del árbol binario de busqueda de enteros en
recorrido por niveles inverso
     */
    @Override
    public String toStringPorNivelesInverso() {
        if(this.raiz != null) return toStringPorNivelesInverso(this.raiz);
        else return "*";
    }
    /**
     * Método que devuelve un String que representa el árbol binario de
busqueda de enteros en recorrido por niveles inverso, comenzando desde el
nodo actual.
     * @param actual el nodo actual desde donde comenzar el recorrido por
niveles inverso
     * @return representación del árbol binario de busqueda de enteros en
recorrido por niveles inverso
     */
    private String toStringPorNivelesInverso(NodoABB actual){
        ArrayCola<NodoABB> q = new ArrayCola<NodoABB>();
        q.encolar(actual); String res = "";
        while(!q.esVacia()){
            NodoABB nodoActual = q.desencolar();
            res += nodoActual.dato.toString()+"\n";
            if(nodoActual.der != null) q.encolar(nodoActual.der);
            if(nodoActual.izq != null) q.encolar(nodoActual.izq);
        }
        return res;
    }
    /**
     * Método que transforma todos los elementos del árbol binario de
busqueda de enteros a sus correspondientes valores negativos.
     * Si el árbol binario de busqueda de enteros está vacío, se lanza una
excepción ABBVacio.
     */
    @Override
    public void transformarABBEnteros() {
        if(this.raiz != null) transformarABBEnteros(this.raiz);
        else try {
            throw new ABBVacio("El árbol esta vacío.");
        } catch (ABBVacio ex) {
```

```
            Logger.getLogger(ABBEnteros.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }
    /**
     * Método que transforma el elemento del nodo actual a su
correspondiente valor negativo,
     * y se realiza la transformación recursivamente en los nodos hijos.
     * @param actual el nodo actual a transformar
     */
    private void transformarABBEnteros(NodoABB actual){
        if(actual != null){
            actual.dato = (int) actual.dato * -1;
        }

        if(actual.izq != null) transformarABBEnteros(actual.izq);
        if(actual.der != null) transformarABBEnteros(actual.der);
    }
    /**
     * Método que realiza la suma de las claves menores que un número dado
en un árbol binario de búsqueda.
     * Solicita al usuario que introduzca una clave y realiza la suma
utilizando un método auxiliar.
     * Imprime el resultado de la suma por pantalla.
     */
    @Override
    public void sumarClavesMinNumDado() {
        System.out.println("Introduzca una clave: ");
        int clave = MyInput.readInt();

        int suma = sumarClavesMinNumDado(raiz, clave);
        System.out.println("La suma de las claves menores que " + clave +"
es: " + suma);
    }
    /**
     * Método auxiliar recursivo que realiza la suma de las claves menores
que un número dado en un subárbol.
     * @param raiz la raíz del subárbol
     * @param num el número dado
     * @return la suma de las claves menores que el número dado en el
subárbol
     */
    private int sumarClavesMinNumDado(NodoABB raiz, int num){
        if(raiz == null){
            return 0;
        }
        else{
            if((int) raiz.dato > num){
                return sumarClavesMinNumDado(raiz.izq, num);
            }
```

```
            else{
                return sumarClavesMinNumDado(raiz.izq, num) +
sumarClavesMinNumDado(raiz.der, num) + (int) raiz.dato;
            }
        }
    }
    /**
     * Método que calcula el antecesor de un nodo en el árbol binario de
busqueda de enteros. Se solicita al usuario que ingrese la clave del nodo.
     */
    @Override
    public void calcularAntecesorNodo() {
        System.out.print("Introduce la clave del nodo: ");
        int clave = MyInput.readInt();

        NodoABB nodo = buscarNodo(raiz, clave);
        if (nodo == null) {
            System.out.println("Esta clave no existe en el árbol.");
        } else {
            NodoABB antecesor = encontrarAntecesor(raiz, nodo);
            if (antecesor == null) {
                System.out.println("El nodo " + clave + " es la raíz del
árbol.");
            } else {
                System.out.println("El antecesor del nodo " + clave + " es
el " + antecesor.dato);
            }
        }
    }
    /**
     * Método que busca un nodo con una clave dada en el árbol binario de
busqueda de enteros a partir del nodo raíz.
     * @param raiz  el nodo raíz del árbol binario de busqueda de enteros
     * @param clave la clave del nodo a buscar
     * @return el nodo con la clave dada, o null si no se encuentra
     */
    private NodoABB buscarNodo(NodoABB raiz, int clave) {
        if (raiz == null || (int) raiz.dato == clave) {
            return raiz;
        }

        if (clave < (int) raiz.dato) {
            return buscarNodo(raiz.izq, clave);
        } else {
            return buscarNodo(raiz.der, clave);
        }
    }
    /**
     * Método que encuentra el antecesor de un nodo en el árbol a partir del
nodo raíz.
```

```
     * @param raiz el nodo raíz del árbol binario de busqueda de enteros
     * @param nodo el nodo para el cual se busca el antecesor
     * @return el antecesor del nodo, o null si no se encuentra
     */
    private NodoABB encontrarAntecesor(NodoABB raiz, NodoABB nodo) {
        if (raiz == null || nodo == null) {
            return null;
        }

        if (raiz.izq == nodo || raiz.der == nodo) {
            return raiz;
        }

        if ((int) nodo.dato < (int) raiz.dato) {
            return encontrarAntecesor(raiz.izq, nodo);
        } else {
            return encontrarAntecesor(raiz.der, nodo);
        }
    }
    /**
     * Método que muestra las hojas de un árbol binario de búsqueda en
posiciones impares.
     * Comienza el recorrido desde la raíz del árbol y llama a un método
auxiliar para realizar la tarea.
     */
    public void mostrarHojasPosImpares() {
        mostrarHojasPosImpares(raiz, 1);
    }
    /**
     * Método auxiliar recursivo que muestra las hojas de un subárbol en
posiciones impares.
     * @param raiz la raíz del subárbol
     * @param posicion la posición actual del nodo en el árbol
     */
    private void mostrarHojasPosImpares(NodoABB actual, int posicion) {
        if (actual == null) {
            return;
        }

        // Si es una hoja y la posición es impar, muestra la clave
        if (actual.izq == null && actual.der == null && posicion % 2 != 0) {
            System.out.println("Hoja " + (posicion / 2 + 1) + " - Clave " +
actual.dato);
        }

        // Recorre los hijos en orden izquierdo y derecho
        mostrarHojasPosImpares(actual.izq, 2 * posicion);
        mostrarHojasPosImpares(actual.der, 2 * posicion + 1);
    }
}
```

Clase NodoABB:

```
package librerias.estructurasDeDatos.jerarquicos;

/**
 * Esta clase contiene los metodos y atributos del NodoABB
 * @author Raúl Colindres y Alfredo Sobrados
 * @param <E>
 */
public class NodoABB<E> {
    protected E dato;
    protected NodoABB<E> izq, der;
    protected int tamanyo;
    /**
     * Método constructor parametrizado
     * @param dato el dato que se va a almacenar en el nodo
     * @param izquierdo el hijo izquierdo del nodo
     * @param derecho el hijo derecho del nodo
     */
    public NodoABB(E dato, NodoABB<E> izquierdo, NodoABB<E> derecho){
        this.dato = dato; izq = izquierdo; der = derecho;
        this.tamanyo = 1;
        if(izq!=null)tamanyo+=izq.tamanyo;if(der!=null)
tamanyo+=der.tamanyo;
    }
    /**
     * Método constructor parametrizado
     * @param dato el dato que se va a almacenar en el nodo
     */
    public NodoABB(E dato){
        this(dato, null, null);
    }
}
```

Clase ArrayCola:

```
package librerias.estructurasDeDatos.lineales;

import librerias.estructurasDeDatos.modelos.*;

/**
 * Esta clase contiene los metodos y atributos del ArrayCola
 * @author Raúl Colindres y Alfredo Sobrados
 * @param <E>
 */
public class ArrayCola<E> implements Cola<E>{
    protected E elArray[];
    protected int fin, primero, tallaActual;
    protected static final int CAPACIDAD_POR_DEFECTO = 200;
    /**
     * Método constructor sin parametros
```

```
     */
    public ArrayCola(){
        elArray = (E[]) new Object[CAPACIDAD_POR_DEFECTO];
        tallaActual=0;
        primero=0;
        fin = -1;
    }
    /**
     * Método que encola elementos en la cola
     * @param x elemento a encolar en la cola
     */
    @Override
    public void encolar(E x) {
        if( tallaActual==elArray.length) duplicarArray();
        fin=incrementa(fin);
        elArray[fin]=x;
        tallaActual++;
    }
    /**
     * Método que desencola elementos en la cola
     * @return elemento desencolado de la cola
     */
    @Override
    public E desencolar() {
        E elPrimero = elArray[primero];
        primero = incrementa(primero);
        tallaActual--;
        return elPrimero;
    }
    /**
     * Metodo que devuelve el primero de la cola
     * @return el elemento primero de la cola
     */
    @Override
    public E primero() {
        return elArray[primero];
    }
    /**
     * Método que comprueba si la cola esta vacía
     * @return si la cola esta vacía true y si esta llena false
     */
    @Override
    public boolean esVacia() {
        return(tallaActual==0);
    }
    /**
     * Método que incrementa el valor del índice y lo devuelve. Si el índice
incrementado alcanza el tamaño del arreglo, se reinicia a cero.
     * @param indice el índice actual a incrementar
     * @return el nuevo valor del índice después de incrementarlo
```

```
     */
    private int incrementa(int indice) {
        if(++indice==elArray.length)indice=0;
        return indice;
    }
    /**
     * Método que devuelve una representación en forma de cadena de
caracteres del array circular
     * @return una cadena de caracteres que representa el array circular
     */
    @Override
    public String toString(){
        String res="";
        int aux=primero;
        for(int i =0;i<tallaActual;i++, aux=incrementa(aux))
            res += elArray[aux]+"";
        return res;
    }
    /**
     * Método que Duplica el tamaño del array interno utilizado por el array
circular.
     * Se crea un nuevo array con el doble de capacidad y se copian los
elementos existentes en él.
     * El array original se reemplaza por el nuevo array duplicado.
     * El primer y último índice se actualizan acorde a la nueva talla.
     */
    private void duplicarArray() {
        E nuevo[]=(E[]) new Object[elArray.length*2];
        for(int i =0;i<tallaActual;i++, primero=incrementa(primero))
            nuevo[i]=elArray[primero];
        elArray=nuevo;
        primero=0;
        fin=tallaActual - 1;
    }
    /**
     * Método que devuelve la cantidad actual de elementos almacenados en el
array circular.
     * @return la talla actual del array circular
     */
    public int tallaActual() {
        return tallaActual;
    }
}
```

Interface Cola:

```
package librerias.estructurasDeDatos.modelos;

/**
 * Esta interfaz contiene los metodos de la cola
 * @author Raúl Colindres y Alfredo Sobrados
 * @param <E>
 */
public interface Cola<E> {
    public void encolar(E x);
    public E desencolar();
    public E primero();
    public boolean esVacia();
}
```

Interface Modelo_ABB:

```
package librerias.estructurasDeDatos.modelos;

import librerias.excepciones.*;

/**
 * Esta interfaz contiene los métodos del árbol binario de búsqueda
 * @author Raúl Colindres y Alfredo Sobrados
 * @param <E>
 */
public interface Modelo_ABB<E> {
    public E recuperar(E x) throws ElementoNoEncontrado;
    public E recuperarMin();
    public E recuperarMax();
    public void insertarSinDuplicados(E x) throws ElementoDuplicado;
    public void insertarConDuplicados(E x);
    public void eliminar(E x) throws ElementoNoEncontrado;
    public E eliminarMin();
    public int tamanyo();
    public int altura();
    public boolean esVacio();
    public String toStringPreOrden();
    public String toStringPostOrden();
    public String toStringInOrden();
    public String toStringPorNiveles();
}
```

Interface Modelo_ABBEnteros:

```
package librerias.estructurasDeDatos.modelos;

import librerias.excepciones.*;

/**
 * Esta interfaz contiene los métodos del árbol binario de búsqueda de
enteros
 * @author Raúl Colindres y Alfredo Sobrados
 * @param <Integer>
 */
public interface Modelo_ABBEnteros<Integer> extends Modelo_ABB<Integer>{
    @Override
    public Integer recuperar(Integer x) throws ElementoNoEncontrado;
    @Override
    public Integer recuperarMin();
    @Override
    public Integer recuperarMax();
    @Override
    public void insertarSinDuplicados(Integer x) throws ElementoDuplicado;
    @Override
    public void insertarConDuplicados(Integer x);
    @Override
    public void eliminar(Integer x) throws ElementoNoEncontrado;
    @Override
    public Integer eliminarMin();
    @Override
    public int tamanyo();
    @Override
    public int altura();
    @Override
    public boolean esVacio();
    @Override
    public String toStringPreOrden();
    @Override
    public String toStringPostOrden();
    @Override
    public String toStringInOrden();
    public String toStringInOrdenInverso();
    @Override
    public String toStringPorNiveles();
    public String toStringPorNivelesInverso();
    public void transformarABBEnteros();
    public void sumarClavesMinNumDado();
    public void calcularAntecesorNodo();
}
```

Clase ABBVacio:

```
package librerias.excepciones;

/**
 * Esta clase contiene la excepción de usuario ABBVacio
 * @author Raúl Colindres y Alfredo Sobrados
 */
public class ABBVacio extends Exception{
    /**
     * Método constructor parametrizado
     * @param dme mensaje que aparece en la excepción de usuario
     */
    public ABBVacio(String dme){
        super(dme);
    }
}
```

Clase ElementoDuplicado:

```
package librerias.excepciones;

/**
 * Esta clase contiene la excepción de usuario ElementoDuplicado
 * @author Raúl Colindres y Alfredo Sobrados
 */
public class ElementoDuplicado extends Exception{
    /**
     * Método constructor parametrizado
     * @param dme mensaje que aparece en la excepción de usuario
     */
    public ElementoDuplicado(String dme){
        super(dme);
    }
}
```

Clase ElementoNoEncontrado:

```
package librerias.excepciones;

/**
 * Esta clase contiene la excepción de usuario ElementoNoEncontrado
 * @author Raúl Colindres y Alfredo Sobrados
 */
public class ElementoNoEncontrado extends Exception{
    /**
     * Método constructor parametrizado
     * @param dme mensaje que aparece en la excepción de usuario
     */
    public ElementoNoEncontrado(String dme){
        super(dme);
    }
}
```

Clase TADABB (Principal):

```
package tad_abb;

import Menu.MenuPrincipal;

/**
 * Esta es la clase principal del proyecto
 * @author Raúl Colindres y Alfredo Sobrados
 */
public class TADABB {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        MenuPrincipal mp = new MenuPrincipal();
        mp.ejecutar();
    }
}
```

Pruebas de ejecución

Pruebas de ejecución de la aplicación desarrollada

Captura del menú principal:

```
Menú Principal
seleccione una opción:
0. Salir
1. Crear ABB de enteros positivos (Equilibrado)
2. Listado de claves en orden ascendente
3. Listado de claves en orden descendente
4. Listado de claves por niveles (converso)
5. Transformar árbol ABB de enteros
6. Sumar claves menores que un número dado
7. Calcular el antecesor de un nodo
8. Mostrar hojas (posiciones impares)
```

Captura de la opción 1 del menú principal: Crear ABB de enteros positivos (Equilibrado):

```
Crear un ABB(Árbol Binario de Busqueda) de claves enteras equilibrado (MÁX:50)
Introduzca la clave entera que quiere añadir al árbol
8
¿Quieres añadir mas claves al árbol?(S/N):
s
Introduzca la clave entera que quiere añadir al árbol
9
¿Quieres añadir mas claves al árbol?(S/N):
s
Introduzca la clave entera que quiere añadir al árbol
1
¿Quieres añadir mas claves al árbol?(S/N):
s
Introduzca la clave entera que quiere añadir al árbol
2
¿Quieres añadir mas claves al árbol?(S/N):
s
Introduzca la clave entera que quiere añadir al árbol
3
¿Quieres añadir mas claves al árbol?(S/N):
s
Introduzca la clave entera que quiere añadir al árbol
4
¿Quieres añadir mas claves al árbol?(S/N):
n
```

Captura de la opción 2 del menú principal: Listado de claves en orden ascendente:

```
Opción 2: Listado de claves en orden ascendente
Salida en pantalla:
1
2
3
4
8
9
```

Captura de la opción 3 del menú principal: Listado de claves en orden descendente:

```
Opción 3: Listado de claves en orden descendente
Salida en pantalla:
9
8
4
3
2
1
```

Captura de la opción 4 del menú principal: Listado de claves por niveles (converso):

```
Opción 4: Listado de claves por niveles (converso)
Salida en pantalla:
3
8
1
9
4
2
```

Captura de la opción 5 del menú principal: Transformar árbol ABB de enteros:

```
Opción 5: Transformar árbol ABB de enteros
Árbol transformado!!
```

```
Opción 3: Listado de claves en orden descendente
Salida en pantalla:
-9
-8
-4
-3
-2
-1
```

Captura de la opción 6 del menú principal: Sumar claves menores que un número dado:

```
Opción 6: Sumar claves menores que un número dado
Introduzca una clave:
7
La suma de las claves menores que 7 es: 10
```

Captura de la opción 7 del menú principal: Calcular el antecesor de un nodo:

```
Opción 7: Calcular el antecesor de un nodo
Introduce la clave del nodo: 8
El antecesor del nodo 8 es el 3
```

Captura de la opción 8 del menú principal: Mostrar hojas (posiciones impares):

```
Opción 8: Mostrar hojas (posiciones impares)
Hoja 3 - Clave 2
Hoja 4 - Clave 9
```

Captura de la opción 0 del menú principal: Salir (despedida del programa):

```
Menú Principal
seleccione una opción:
0. Salir
1. Crear ABB de enteros positivos (Equilibrado)
2. Listado de claves en orden ascendente
3. Listado de claves en orden descendente
4. Listado de claves por niveles (converso)
5. Transformar árbol ABB de enteros
6. Sumar claves menores que un número dado
7. Calcular el antecesor de un nodo
8. Mostrar hojas (posiciones impares)
0
Gracias por utilizar nuestros TAD ABB y ABBEnteros
```