Contenido

Práctica 2B: TAD AB

[1) Parte 1 1](#_Toc483329436)

[a) Especificación lógica TAD AB 1](#_Toc483329437)

[b) Implementación TAD AB 1](#_Toc483329438)

[2) Parte 2 1](#_Toc483329439)

[c) Implementación ABEnteros 1](#_Toc483329440)

[3) Parte 3 1](#_Toc483329441)

[d) Cambios para ABB 1](#_Toc483329442)

# Parte 1

## Especificación lógica TAD AB

* + 1. Elementos

El tipo de datos que se puede almacenar en la estructura es cualquier tipo de datos deribado de Object en java, teniendo en cuenta que todos los nodos del árbol deberán tener el mismo tipo.

* + 1. Tipo de Organización

La organización de la ED de arbol es jerárquica, distribuidos en niveles.

* + 1. Dominio

El dominio de la estructura es cualquier elemento que derive de la clase Object en java.

* + 1. Operaciones AB
       1. E raiz()
* Nombre: raiz()
* Descripción: Devuelve el elemento almacenado en el nodo raíz del arbol.
* Datos de entrada: ninguno.
* Salida: elemento del nodo raíz.
* Precondiciones: tener un elemento almacenado.
* Postcondiciones: recuperar el elemento almacenado en el nodo raíz.
  + - 1. Boolean esVacio()
* Nombre: esVacio().
* Descripción: indica si un arbol es vacio.
* Datos de entrada: ninguno.
* Salida: booleano que indica si el arbol está vacio.
* Precondiciones: ninguna.
* Postcondiciones: conocer si el árbol está vacio.
  + - 1. ArbolBinario hijoIzq()
* Nombre: hijoIzq()
* Descripción: Devuelve un subarbol con el nodo raíz como el hijo derecho del nodo raíz anterior.
* Datos de entrada: ninguno.
* Salida: EnlazadoArbolBinario hijo izquierdo de la raíz.
* Precondiciones: ninguna.
* Postcondiciones: Obtener el hijo izquierdo.
  + - 1. ArbolBinario hijoDer()
* Nombre: hijoDer().
* Descripción: Devuelve un subarbol con el nodo raíz como el hijo derecho del nodo raíz anterior.
* Datos de entrada: ninguno.
* Salida:EnlazadoArbolBinario hijo derecho del nodo raíz.
* Precondiciones: ninguna.
* Postcondiciones: Obtener el hijo derecho de la raíz.
  + - 1. Boolean esta(E elemento): lanzadera
* Nombre: esta().
* Descripción: Indica si el elemento que se le pasa como parámetro se encuentra en el árbol.
* Datos de entrada: elemento a buscar.
* Salida: booleano que indica si está en el arbol
* Precondiciones: ninguna.
* Postcondiciones: saber si un elemento se encuentra en el arbol.
  + - 1. Boolean esta(NodoBinario<E> raiz, E elemento): recursivo
* Nombre: esta().
* Descripción: Indica si el elemento que se le pasa como parámetro se encuentra en el árbol.
* Datos de entrada: nodoBinario raiz y elemento.
* Salida: booleano que indica si el elemento está o no en el arbol.
* Precondiciones: ejecutar el metodo lanzadera.
* Postcondiciones: conocer si el elemento se encuentra en el arbol.
  + - 1. Void setHijoIzq(ArbolBinario hi)
* Nombre: setHijoIzq().
* Descripción: Introducir el hijo izquierdo del nodo raíz.
* Datos de entrada: ArbolBinario hi.
* Salida: ninguna.
* Precondiciones: ninguno.
* Postcondiciones: tener insertado en el arbol como hijo izquierdo el arbol introducido como parámetro.
  + - 1. Void setHijoDer(ArbolBinario hd)
* Nombre: setHijoDer().
* Descripción: Introducir el hijo derecho del nodo raíz.
* Datos de entrada: ArbolBinario hijo derecho.
* Salida: ninguna
* Precondiciones: ninguna.
* Postcondiciones: tener insertado en el arbol como hijo derecho el arbol introducido como parámetro.
  + - 1. Void suprimir()
* Nombre: suprimir()
* Descripción: pone el nodo raíz a null.
* Datos de entrada: ninguno.
* Salida: ninguna.
* Precondiciones: ninguna.
* Postcondiciones: el nodo raíz del arbol ahora es null, perdemos todos los nodos que cuelgan de él.
  + - 1. Void EliminarNodosInferiores(int nivel): lanzadera
* Nombre: EliminarNodosInferiores()
* Descripción: Elimina los nodos inferiores al nivel introducido como parámetro de forma recursiva.
* Datos de entrada: int nivel
* Salida: ninguna
* Precondiciones: ninguna.
* Postcondiciones: Los nodo inferiores al nivel introducido como parámetro han sido eliminados.
  + - 1. Void EliminarNodosInferiores(ArbolBinario<E> a, int cont, int nivel): recursivo
* Nombre: EliminarNodosInferiores().
* Descripción: Elimina los nodos inferiores al nivel introducido como parámetro de forma recursiva.
* Datos de entrada: ArbolBinario<E> a, int cont, int nivel.
* Salida: ninguna
* Precondiciones: ninguna.
* Postcondiciones: Los nodo inferiores al nivel introducido como parámetro han sido eliminados.
  + - 1. Void setRaiz(E elemRaiz)
* Nombre: setRaiz().
* Descripción: Introduce al nodo raíz el elemento introducido como parámetro.
* Datos de entrada: E elemento
* Salida: ninguna.
* Precondiciones: ninguna.
* Postcondiciones: el nodo raíz guarda el el elemento pasado como parámetro.
  + - 1. Void anchura(ArbolBinario<E> a)
* Nombre: anchura().
* Descripción: Recorre el árbol en anchura de forma recursiva.
* Datos de entrada: ArbolBinario<E> a.
* Salida: ninguna.
* Precondiciones: ninguna.
* Postcondiciones: conocemos la secuencia de los nodos por niveles.
  + - 1. Void preorden(ArbolBinario<E> a)
* Nombre: preorden().
* Descripción: recorrido del arbol en preorden.
* Datos de entrada: ArbolBinario<E> a.
* Salida: ninguna.
* Precondiciones: ninguna.
* Postcondiciones: conocemos la secuencia de los nodos por un recorrido en preorden.
  + - 1. Void inorden(ArbolBinario<E> a)
* Nombre: inorden().
* Descripción: Recorrido del arbol en inorden.
* Datos de entrada: ArbolBinario<E> a.
* Salida: ninguna.
* Precondiciones: ninguna.
* Postcondiciones: conocemos la secuencia de los nodos por un recorrido en inorden.
  + - 1. Void postorden(ArbolBinario<E> a)
* Nombre: postorden().
* Descripción: recorrido del árbol en postorden.
* Datos de entrada: ArbolBinario<E> a.
* Salida: ninguna.
* Precondiciones: ninguna.
* Postcondiciones: conocemos la secuencia de los nodos por un recorrido en inorden.

/\*

* + 1. Operaciones de la Clase ABEnteros que extiende a EnlazadoArbolBinario
       1. Boolean esABB()
* Nombre: esABB().
* Descripción: comprueba que los nodos del arbol estén ordenados de tal forma que se corresponda con un arbol binario de búsqueda.
* Datos de entrada: ninguno
* Salida: ninguna.
* Precondiciones: ninguna.
* Postcondiciones:

\*/

## Implementación TAD AB

//código EnlazadoArbolBinario, NodoBinario, e interfáz

# Parte 2

## Implementación ABEnteros

//código ABEnteros

# Parte 3

## Cambios para ABB

Si hubiesemos implementado los métodos pedidos para un arbol binario de búsqueda, las modificaciones serían las siguientes:

* + 1. EliminarNodosInferiores()

Este método quedaría igual, puesto que para eliminar los nodos inferiores a un nivel, solo tenemos en cuenta los niveles, y no los valores de los nodos, por lo tanto no importa que el árbol esté ordenado o no (sea binario de busqueda) para está operación.

* + 1. CrearAB1() y AB2()

Para crear un arbol binario de busqueda, los nodos no se pueden insertar donde sea, por lo tanto al insertar nodos, en el método del arbol se debe comparar los valores de los nodos a insertar con su nodo padre para insertarlo como hijo izquierdo o derecho en función de si es menor o mayor, y así crear el arbol ordenado.

* + 1. EsABB()

Al crear un árbol binario de búsqueda, he ir comparando en el constructor los valores para ordenar el arbol, este método carece de sentido, puesto que siempre es un arbol binario de búsqueda.

Nota: Hay que tener en cuenta a la hora de eliminar nodos que el arbol tiene que permanecer ordenado, reasignando enlaces.

* + 1. RaizIgualNodosInternos()

Para este método, al no importarnos el valor de los nodos, y solo el numero de nodos que hay, no sería necesario hacer modificaciones.

Se podria añadir una condición para que una vez que el contador supere el valor del nodo raíz, aborte la ejecución puesto que aunque sigua ejecutando, el valor del contador va a ser superior siempre.

* + 1. MinimoValorNivel()