

**Estructuras de Datos y Algoritmos I**  
**Grado en Ingeniería Informática, Curso 2º**

**ACTIVIDAD 3**

**Implementación de operaciones sobre ABBs**

**Objetivos**

- Aprender a utilizar árboles binarios de búsqueda (ABB) para tener un conjunto de datos organizados de tal forma, que cuando se deseen obtener puedan estar ordenados según una clave y una relación de orden.
- Aprender ABB balanceados como son el AVL y el Rojo-Negro. Realizar comparativas con ellos con el objetivo de ver cuál es el más apropiado dependiendo del caso.
- Aprender a añadir funcionalidad adicional a un ABB y a un AVL.
- Aprender a ‘colgar’ una nueva estructura de datos (ABB AVL) de la JCF, debiendo implementar todas las funciones que tienen que realizar dependiendo de qué interfaz extienda o de qué clase derive.

**Requerimientos**

Para superar esta actividad se debe realizar lo siguiente:

- Dominar cómo añadir funcionalidad a un ABB y a un AVL.
- Dominar cómo comparar varias estructuras de datos arbóreas como son ABB, AVL y Rojo-Negro, extrayendo las conclusiones correspondientes.
- Dominar cómo ‘colgar’ de la JCF una nueva estructura de datos como es el ABB AVL y sabiendo cómo implementar toda la funcionalidad adicional que se requiere.

**Enunciado**

En esta actividad vamos a complementar lo estudiado en las clases de teoría, con la utilización de los árboles binarios de búsqueda (BSTree) y ABB balanceados AVL (AVLTree) en determinados problemas, junto con una comparativa con los ABB balanceados Rojo-Negros (RBTree). Para ello, se utilizará las implementaciones base suministradas en el paquete `org.eda.estructurasdedatos`, que se deberán extender convenientemente con las funciones demandadas.

**Ejercicio 1.** Implementar un nuevo método en la clase **BSTree** que calcule la altura del árbol, en su versión recursiva.

```
private int height(BSTNode<T> t) {  
    // ...  
}  
  
public int height() {  
    return height(this.root);  
}
```

**Ejercicio 2.** Implementar un nuevo método en la clase **BSTree** que obtenga en número total de hojas que tiene el árbol, en su versión recursiva.

```
public int numberOfLeaves() {  
    return numberOfLeaves(root);  
}  
  
private int numberOfLeaves(BSTNode<T> t) {  
    // ...  
}
```

**Ejercicio 3.** Implementar dos nuevos métodos en la clase **BSTree** que encuentre el mínimo y el máximo de los elementos que tiene el árbol (si éste no está vacío), tanto en su versión recursiva como iterativa.

```
public T findMin() {  
    return findMin(root).nodeValue;  
}  
  
private BSTNode<T> findMin(BSTNode<T> t) {  
    / ...  
}  
  
public T findMinIterative() {  
    return findMinIterative(root).nodeValue;  
}  
  
private BSTNode<T> findMinIterative(BSTNode<T> t) {  
    // ...  
}  
  
public T findMax() {  
    return findMax(root).nodeValue;  
}  
  
private BSTNode<T> findMax(BSTNode<T> t) {  
    // ...  
}
```

```
public T findMaxIterative() {  
    return findMaxIterative(root).nodeValue;  
}  
  
private BSTNode<T> findMaxIterative(BSTNode<T> t) {  
    // ...  
}
```

**Ejercicio 4.** Implementar una nueva función (preferentemente recursiva) en la clase **BSTree** que, devuelva la información de los nodos de un determinado nivel, proporcionado como parámetro.

```
public String toStringLevel(int level) {  
    // ...  
}  
  
private String toStringLevel(BSTNode<T> t, int level) {  
    // ...  
}
```

**Ejercicio 5.** Implementar la versión iterativa del método `toStringInorder()` de clase **BSTree**, comprobando su correcto funcionamiento con la versión recursiva.

```
public String toStringIterativeInorder() {  
    String s = toStringIterativeInorder(root);  
    return s;  
}  
  
public static <T> String toStringIterativeInorder(BSTNode<T> t) {  
    // ...  
}
```

**Ejercicio 6.** Implementar la versión iterativa del método `toStringPostorder()` de clase **BSTree**, comprobando su correcto funcionamiento con la versión recursiva.

```
public String toStringIterativePostorder() {  
    String s = toStringIterativePostorder(root);  
    return s;  
}  
  
public static <T> String toStringIterativePostorder(BSTNode<T> t) {  
    // ...  
}
```

**Ejercicio 7.** Implementar un nuevo método para la clase **BSTree** que, partiendo de un **BSTree** vacío, inserte los datos (por ejemplo `Integer`) almacenados en un `ArrayList` de forma que el árbol binario de búsqueda resultante quede equilibrado.

```
public void addBalanced(ArrayList<T> aL) {  
    // ...  
    addBalanced(aL, 0, aL.size() - 1);  
}  
  
private void addBalanced(ArrayList<T> aL, int left, int right) {  
    // ...  
}
```

**Ejercicio 8.** Implementar una nueva función para la clase **BSTree** que, devuelva un ABB simétrico a un ABB dado. Como aclaración, un ABB simétrico es el que se construye a partir de uno dado, convirtiendo el subárbol izquierdo en subárbol derecho, y viceversa.

```
public BSTNode<T> buildSimetricTree() {  
    // ...  
}  
  
private BSTNode<T> buildSimetricTree(BSTNode<T> t) {  
    // ...  
}
```

**Ejercicio 9.** Implementar una nueva función para la clase **BSTree** que, devuelva la profundidad desde la raíz a la que se encuentra un nodo que contiene un determinado valor de clave **x**. Si dicho valor no se encuentra en el árbol, la función deberá devolver -1. La implementación puede ser tanto recursiva como iterativa (no recursiva).

```
public int pathHeight(T x) {  
    // ...  
}
```

**Ejercicio 10.** Dada la estructura de datos **AVLTree** (implementada, casi totalmente, en el libro de Topp-Ford), añadir los métodos necesarios para realizar las operaciones de **writeObject** (serialización) y **readObject** (deserialización). Es decir, implementar las operaciones `writeObject` y `readObject`, siguiendo el mismo esquema presentado para las colecciones de la Actividad 01, para que puedan serializar todos los datos gestionados por dicha estructura de datos (**AVLTree**).

```
public class AVLTree<T> implements ..., Serializable {

    // Code for the class

    private void writeObject(ObjectOutputStream out) throws java.io.IOException {

        out.defaultWriteObject();

        // write out the necessary elements of the collection (out.writeObject(...))

    }

    private void readObject(ObjectInputStream in) throws IOException,
                                                                    ClassNotFoundException {

        in.defaultReadObject();

        // read the elements of the collection (using in.readObject())

    }
    // Code for the class
}
```

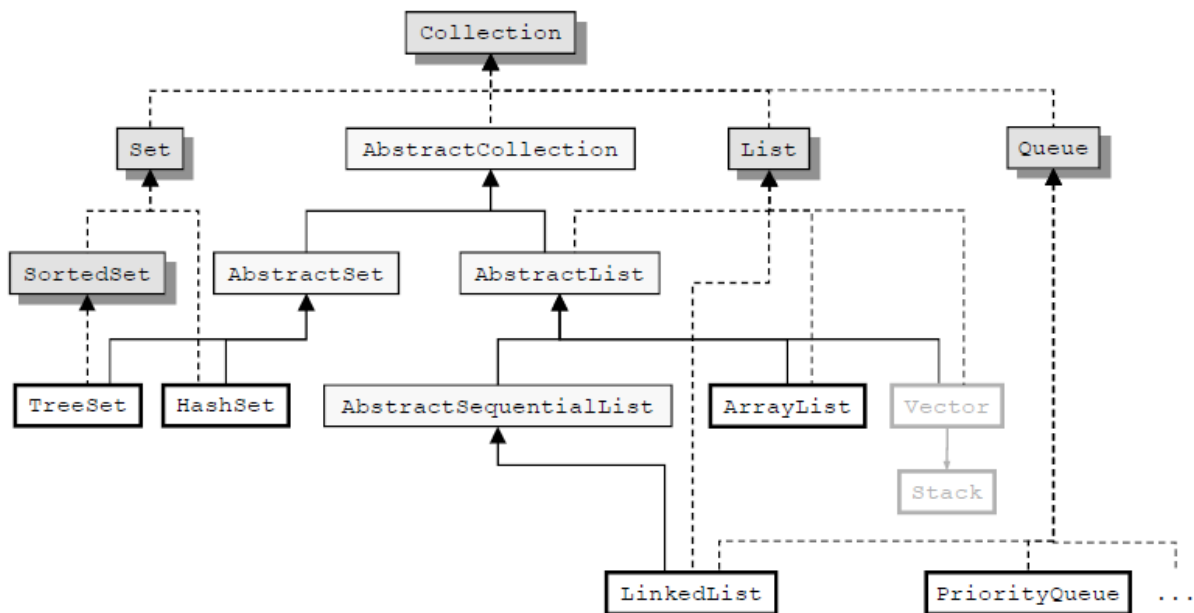
**Ejercicio 11.** Implementar tanto en el AVLTree como en el RBTree (ya está implementada en el ejercicio 10 para BSTree) la función `pathHeight`, que devuelve la profundidad desde la raíz a la que se encuentra un nodo que contiene un determinado valor de clave **x**. Si dicho valor no se encuentra en el árbol, la función deberá devolver -1. La implementación de dicha función, como ya se indicó en el ejercicio 10, puede ser tanto recursiva como iterativa (no recursiva).

```
public int pathHeight(T x) {
    // ...
}
```

Para comprobar el rendimiento de los árboles binarios de búsqueda en cuestión (BSTree, AVLTree y RBTree), se propone un test que crea un ArrayList con 100000 de Integer diferentes y baraja dicha colección (`shuffle`). Luego inserta dichos elementos en cada tipo de ABB. A continuación procesa el ArrayList y para cada elemento llama a la función `pathHeight` de su respectivo ABB, manteniendo un contador de la profundidad acumulada de los elementos desde la raíz. Finalmente, se calcula la profundidad media (*average*) de un elemento en cada uno de los tres ABBs.

**Ejercicio 12.** Implementar que nuestro **AVLTree** cuelgue de la JCF, en concreto de **AbstractSet** (clase de la JCF). Además de las operaciones que ya tenemos implementadas: **AVLTree**, **add**, **remove**, **clear**, **contains**, **isEmpty**, **iterator**, **size**, **toArray** y **toString**, se deben implementar las siguientes operaciones, verificando el test correspondiente.

```
public AVLTree(AVLTree<T> otherTree) { ... }
public boolean equals (Object obj) { ... }
public boolean removeAll(Collection c) { ... }
public boolean addAll(Collection c) { ... }
public boolean containsAll(Collection c) { ... }
public boolean retainAll(Collection c) { ... }
```



Aclaraciones sobre qué deben hacer cada una de las nuevas funciones a implementar en el **AVLTree** (**AbstractSet** <http://docs.oracle.com/javase/1.4.2/docs/api/java/util/AbstractSet.html>):

```
public AVLTree(AVLTree<T> otherTree) { ... }
```

Constructor copia de un **AVLTree**.

```
public boolean equals (Object obj) { ... }
```

Comprueba si dos **AVLTrees** son iguales en contenido y en estructura.

```
public boolean removeAll(Collection c) { ... }
```

Elimina del **AVLTree** todos los elementos que están contenidos en la colección **c**.

```
public boolean addAll(Collection c) { ... }
```

Añade al **AVLTree** todos los elementos contenidos en la colección **c** y que evidentemente no están en dicho **AVLTree**.

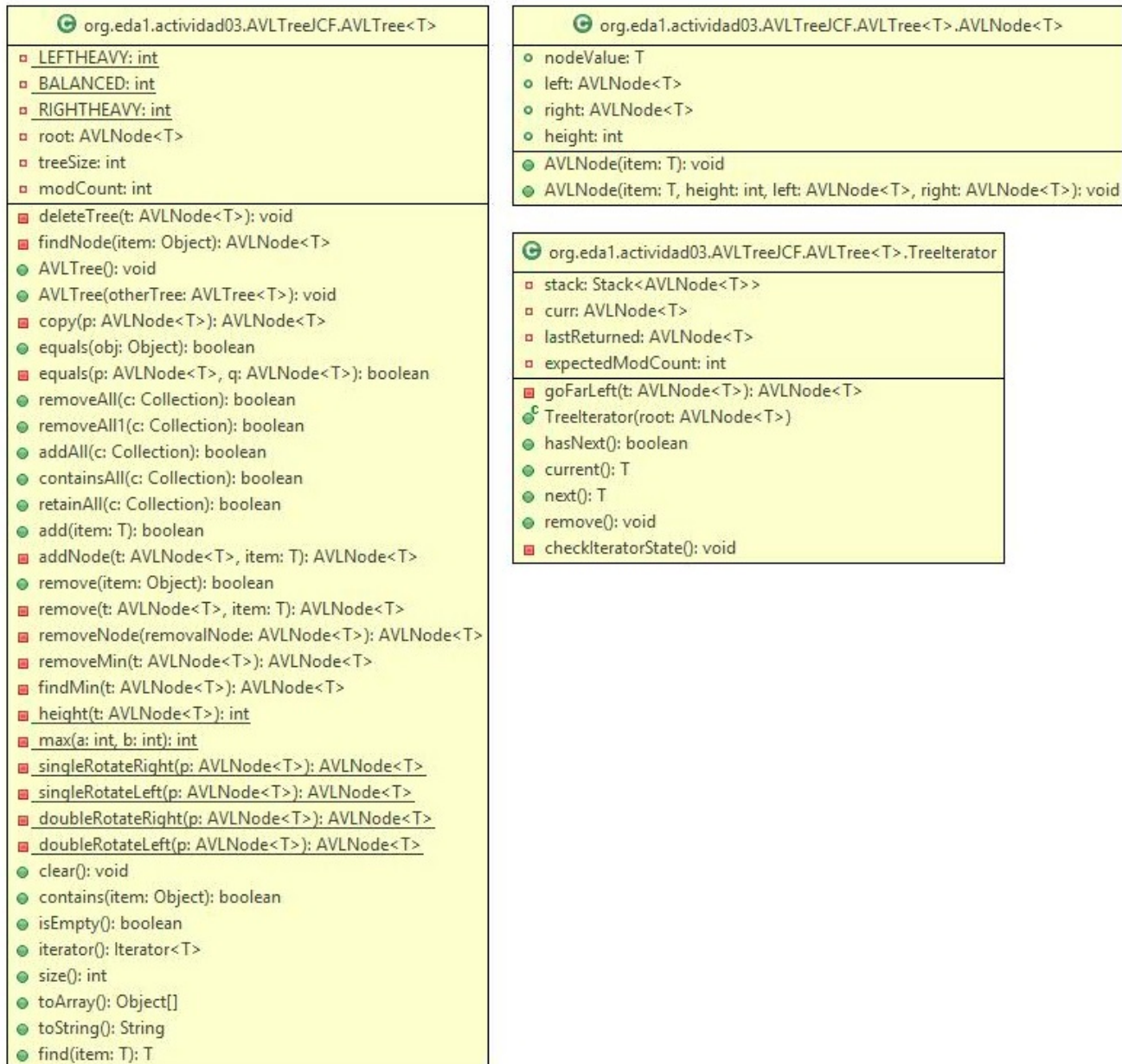
```
public boolean containsAll(Collection c) { ... }
```


Devuelve true si el AVLTree contiene TODOS los elementos contenidos en la colección *c*.


```
public boolean retainAll(Collection c) { ... }
```


Conserva sólo los elementos del AVLTree que están contenidos en la colección *c*. Es decir, se eliminan del AVLTree todos los elementos que no están contenidos en la colección *c*.

A continuación se exponen todos los diagramas de clases UML para cada una de las clases que serán necesarias para el desarrollo de todos los ejercicios de esta actividad.





 org.eda1.actividad03.BSTree.BSTree<T>
<ul style="list-style-type: none"> <li>root: BSTNode&lt;T&gt;</li> <li>treeSize: int</li> <li>modCount: int</li> <li>n2: int</li> </ul>
<ul style="list-style-type: none"> <li>removeNode(dNode: BSTNode&lt;T&gt;): void</li> <li>findNode(item: Object): BSTNode&lt;T&gt;</li> <li>BSTree(): void</li> <li>add(item: T): boolean</li> <li>clear(): void</li> <li>contains(item: Object): boolean</li> <li>isEmpty(): boolean</li> <li>iterator(): Iterator&lt;T&gt;</li> <li>remove(item: Object): boolean</li> <li>size(): int</li> <li>toArray(): Object[]</li> <li>toString(): String</li> <li>find(item: T): T</li> <li>clone(): Object</li> <li>toStringPreorder(t: BSTNode&lt;T&gt;): String</li> <li>toStringPreorder(): String</li> <li>toStringInorder(t: BSTNode&lt;T&gt;): String</li> <li>toStringInorder(): String</li> <li>toStringPostorder(t: BSTNode&lt;T&gt;): String</li> <li>toStringPostorder(): String</li> <li>toStringBreadthFirstTraversal(t: BSTNode&lt;T&gt;): String</li> <li>toStringBreadthFirstTraversal(): String</li> <li>toStringIterativePreorder(t: BSTNode&lt;T&gt;): String</li> <li>toStringIterativePreorder(): String</li> <li>toStringIterativeInorder(t: BSTNode&lt;T&gt;): String</li> <li>toStringIterativeInorder(): String</li> <li>toStringIterativePostorder(t: BSTNode&lt;T&gt;): String</li> <li>toStringIterativePostorder(): String</li> <li>toStringIterativePreorderEasy(t: BSTNode&lt;T&gt;): String</li> <li>toStringIterativePreorderEasy(): String</li> <li>toStringIterativeInorderEasy(t: BSTNode&lt;T&gt;): String</li> <li>toStringIterativeInorderEasy(): String</li> <li>toStringIterativePostorderEasy(t: BSTNode&lt;T&gt;): String</li> <li>toStringIterativePostorderEasy(): String</li> <li>addBalanced(aL: ArrayList&lt;T&gt;): void</li> <li>addBalanced(aL: ArrayList&lt;T&gt;, left: int, right: int): void</li> <li>height(t: BSTNode&lt;T&gt;): int</li> <li>height(): int</li> <li>numberOfLeaves(): int</li> <li>numberOfLeaves(t: BSTNode&lt;T&gt;): int</li> <li>findMin(): T</li> <li>findMin(t: BSTNode&lt;T&gt;): BSTNode&lt;T&gt;</li> <li>findMinIterative(): T</li> <li>findMinIterative(t: BSTNode&lt;T&gt;): BSTNode&lt;T&gt;</li> <li>findMax(): T</li> <li>findMax(t: BSTNode&lt;T&gt;): BSTNode&lt;T&gt;</li> <li>findMaxIterative(): T</li> <li>findMaxIterative(t: BSTNode&lt;T&gt;): BSTNode&lt;T&gt;</li> <li>pathHeight(item: T): int</li> <li>toStringLevel(level: int): String</li> <li>toStringLevel(t: BSTNode&lt;T&gt;, level: int): String</li> <li>toStringLevel2(level: int): String</li> <li>toStringLevel2(t: BSTNode&lt;T&gt;, level: int): String</li> <li>findLevel(curr: BSTNode&lt;T&gt;, x: T): int</li> <li>findLevel(item: T): int</li> <li>equal(t: BSTree&lt;T&gt;): boolean</li> </ul>


 org.eda1.actividad03.BSTree.BSTree<T>.Treeliterator
<ul style="list-style-type: none"> <li>expectedModCount: int</li> <li>lastReturned: BSTNode&lt;T&gt;</li> <li>nextNode: BSTNode&lt;T&gt;</li> </ul>
<ul style="list-style-type: none"> <li>Treeliterator()</li> <li>hasNext(): boolean</li> <li>current(): T</li> <li>next(): T</li> <li>remove(): void</li> <li>checkIteratorState(): void</li> </ul>


 org.eda1.actividad03.BSTree.BSTree<T>.BSTNode<T>
<ul style="list-style-type: none"> <li>nodeValue: T</li> <li>left: BSTNode&lt;T&gt;</li> <li>right: BSTNode&lt;T&gt;</li> <li>parent: BSTNode&lt;T&gt;</li> </ul>
<ul style="list-style-type: none"> <li>BSTNode(item: T, parentNode: BSTNode&lt;T&gt;): void</li> </ul>





 org.eda1.actividad03.BS_AVL_RB_Tree.RBTree<T>
<ul style="list-style-type: none"> <li>▣ treeSize: int</li> <li>▣ root: RBNode&lt;T&gt;</li> <li>▣ modCount: int</li> <li>▣ NIL: RBNode&lt;T&gt;</li> </ul>
<ul style="list-style-type: none"> <li>▣ deleteTree(t: RBNode&lt;T&gt;): void</li> <li>▣ removeNode(dNode: RBNode&lt;T&gt;): void</li> <li>▣ findNode(item: Object): RBNode&lt;T&gt;</li> <li>▣ makeEmptyTree(): void</li> <li>▣ rbDeleteFixup(x: RBNode&lt;T&gt;): void</li> <li>▣ rotateLeft(pivot: RBNode&lt;T&gt;): void</li> <li>▣ rotateRight(pivot: RBNode&lt;T&gt;): void</li> <li>▣ split4Node(x: RBNode&lt;T&gt;): void</li> <li>● RBTree(): void</li> <li>● add(item: T): boolean</li> <li>● clear(): void</li> <li>● contains(item: Object): boolean</li> <li>● isEmpty(): boolean</li> <li>● iterator(): Iterator&lt;T&gt;</li> <li>● remove(item: Object): boolean</li> <li>● size(): int</li> <li>● toArray(): Object[]</li> <li>● toString(): String</li> <li>▣ height(t: RBNode&lt;T&gt;): int</li> <li>● height(): int</li> <li>● pathHeight(item: T): int</li> </ul>


 org.eda1.actividad03.BS_AVL_RB_Tree.RBTree<T>.Treeliterator
<ul style="list-style-type: none"> <li>▣ expectedModCount: int</li> <li>▣ lastReturned: RBNode&lt;T&gt;</li> <li>▣ nextNode: RBNode&lt;T&gt;</li> </ul>
<ul style="list-style-type: none"> <li>▲ Treeliterator()</li> <li>● hasNext(): boolean</li> <li>● current(): T</li> <li>● next(): T</li> <li>● remove(): void</li> <li>▣ checkIteratorState(): void</li> </ul>


 org.eda1.actividad03.BS_AVL_RB_Tree.RBTree<T>.RBNode<T>
<ul style="list-style-type: none"> <li>● BLACK: int</li> <li>● RED: int</li> <li>● parent: RBNode&lt;T&gt;</li> <li>● left: RBNode&lt;T&gt;</li> <li>● right: RBNode&lt;T&gt;</li> <li>● color: int</li> <li>● nodeValue: T</li> </ul>
<ul style="list-style-type: none"> <li>● RBNode(item: T, left: RBNode&lt;T&gt;, right: RBNode&lt;T&gt;, parent: RBNode&lt;T&gt;, color: int): void</li> </ul>


 org.eda1.actividad03.BS_AVL_RB_Tree.AVLTree<T>
<ul style="list-style-type: none"> <li>root: AVLNode&lt;T&gt;</li> <li>treeSize: int</li> <li>modCount: int</li> </ul>
<ul style="list-style-type: none"> <li>deleteTree(t: AVLNode&lt;T&gt;): void</li> <li>findNode(item: Object): AVLNode&lt;T&gt;</li> <li>AVLTree(): void</li> <li>add(item: T): boolean</li> <li>addNode(t: AVLNode&lt;T&gt;, item: T): AVLNode&lt;T&gt;</li> <li>remove(item: Object): boolean</li> <li>remove(t: AVLNode&lt;T&gt;, item: T): AVLNode&lt;T&gt;</li> <li>removeNode(removalNode: AVLNode&lt;T&gt;): AVLNode&lt;T&gt;</li> <li>removeMin(t: AVLNode&lt;T&gt;): AVLNode&lt;T&gt;</li> <li>findMin(t: AVLNode&lt;T&gt;): AVLNode&lt;T&gt;</li> <li>height(t: AVLNode&lt;T&gt;): int</li> <li>max(a: int, b: int): int</li> <li>singleRotateRight(p: AVLNode&lt;T&gt;): AVLNode&lt;T&gt;</li> <li>singleRotateLeft(p: AVLNode&lt;T&gt;): AVLNode&lt;T&gt;</li> <li>doubleRotateRight(p: AVLNode&lt;T&gt;): AVLNode&lt;T&gt;</li> <li>doubleRotateLeft(p: AVLNode&lt;T&gt;): AVLNode&lt;T&gt;</li> <li>clear(): void</li> <li>contains(item: Object): boolean</li> <li>isEmpty(): boolean</li> <li>iterator(): Iterator&lt;T&gt;</li> <li>size(): int</li> <li>toArray(): Object[]</li> <li>toString(): String</li> <li>find(item: T): T</li> <li>height(): int</li> <li>pathHeight(item: T): int</li> </ul>

 org.eda1.actividad03.BS_AVL_RB_Tree.AVLTree<T>.AVLNode<T>
<ul style="list-style-type: none"> <li>nodeValue: T</li> <li>left: AVLNode&lt;T&gt;</li> <li>right: AVLNode&lt;T&gt;</li> <li>height: int</li> </ul>
<ul style="list-style-type: none"> <li>AVLNode(item: T): void</li> </ul>

 org.eda1.actividad03.BS_AVL_RB_Tree.AVLTree<T>.Treeliterator
<ul style="list-style-type: none"> <li>stack: ArrayList&lt;AVLNode&lt;T&gt;&gt;</li> <li>curr: AVLNode&lt;T&gt;</li> <li>lastReturned: AVLNode&lt;T&gt;</li> <li>expectedModCount: int</li> </ul>
<ul style="list-style-type: none"> <li>goFarLeft(t: AVLNode&lt;T&gt;): AVLNode&lt;T&gt;</li> <li>Treeliterator(root: AVLNode&lt;T&gt;)</li> <li>hasNext(): boolean</li> <li>current(): T</li> <li>next(): T</li> <li>remove(): void</li> <li>checkIteratorState(): void</li> </ul>

 org.eda1.actividad03.AVLTree.AVLTree<T>
<ul style="list-style-type: none"> <li>root: AVLNode&lt;T&gt;</li> <li>treeSize: int</li> <li>modCount: int</li> </ul>
<ul style="list-style-type: none"> <li>deleteTree(t: AVLNode&lt;T&gt;): void</li> <li>findNode(item: Object): AVLNode&lt;T&gt;</li> <li>AVLTree(): void</li> <li>add(item: T): boolean</li> <li>addNode(t: AVLNode&lt;T&gt;, item: T): AVLNode&lt;T&gt;</li> <li>remove(item: Object): boolean</li> <li>remove(t: AVLNode&lt;T&gt;, item: T): AVLNode&lt;T&gt;</li> <li>removeNode(removalNode: AVLNode&lt;T&gt;): AVLNode&lt;T&gt;</li> <li>removeMin(t: AVLNode&lt;T&gt;): AVLNode&lt;T&gt;</li> <li>findMin(t: AVLNode&lt;T&gt;): AVLNode&lt;T&gt;</li> <li>height(t: AVLNode&lt;T&gt;): int</li> <li>max(a: int, b: int): int</li> <li>singleRotateRight(p: AVLNode&lt;T&gt;): AVLNode&lt;T&gt;</li> <li>singleRotateLeft(p: AVLNode&lt;T&gt;): AVLNode&lt;T&gt;</li> <li>doubleRotateRight(p: AVLNode&lt;T&gt;): AVLNode&lt;T&gt;</li> <li>doubleRotateLeft(p: AVLNode&lt;T&gt;): AVLNode&lt;T&gt;</li> <li>clear(): void</li> <li>contains(item: Object): boolean</li> <li>isEmpty(): boolean</li> <li>iterator(): Iterator&lt;T&gt;</li> <li>size(): int</li> <li>toArray(): Object[]</li> <li>toString(): String</li> <li>find(item: T): T</li> <li>height(): int</li> <li>pathHeight(item: T): int</li> <li>writeObject(out: java.io.ObjectOutputStream): void</li> <li>readObject(in: java.io.ObjectInputStream): void</li> </ul>

 org.eda1.actividad03.AVLTree.AVLTree<T>.Treeliterator
<ul style="list-style-type: none"> <li>stack: ALStack&lt;AVLNode&lt;T&gt;&gt;</li> <li>curr: AVLNode&lt;T&gt;</li> <li>lastReturned: AVLNode&lt;T&gt;</li> <li>expectedModCount: int</li> </ul>
<ul style="list-style-type: none"> <li>goFarLeft(t: AVLNode&lt;T&gt;): AVLNode&lt;T&gt;</li> <li>Treeliterator(root: AVLNode&lt;T&gt;)</li> <li>hasNext(): boolean</li> <li>current(): T</li> <li>next(): T</li> <li>remove(): void</li> <li>checkIteratorState(): void</li> </ul>

 org.eda1.actividad03.AVLTree.AVLTree<T>.AVLNode<T>
<ul style="list-style-type: none"> <li>nodeValue: T</li> <li>left: AVLNode&lt;T&gt;</li> <li>right: AVLNode&lt;T&gt;</li> <li>height: int</li> </ul>
<ul style="list-style-type: none"> <li>AVLNode(item: T): void</li> </ul>

org.eda1.actividad03.BS_AVL_RB_Tree.RBTree<T>
<ul style="list-style-type: none"> <li>treeSize: int</li> <li>root: RBNode&lt;T&gt;</li> <li>modCount: int</li> <li>NIL: RBNode&lt;T&gt;</li> </ul>
<ul style="list-style-type: none"> <li>deleteTree(t: RBNode&lt;T&gt;): void</li> <li>removeNode(dNode: RBNode&lt;T&gt;): void</li> <li>findNode(item: Object): RBNode&lt;T&gt;</li> <li>makeEmptyTree(): void</li> <li>rbDeleteFixup(x: RBNode&lt;T&gt;): void</li> <li>rotateLeft(pivot: RBNode&lt;T&gt;): void</li> <li>rotateRight(pivot: RBNode&lt;T&gt;): void</li> <li>split4Node(x: RBNode&lt;T&gt;): void</li> <li>RBTree(): void</li> <li>add(item: T): boolean</li> <li>clear(): void</li> <li>contains(item: Object): boolean</li> <li>isEmpty(): boolean</li> <li>iterator(): Iterator&lt;T&gt;</li> <li>remove(item: Object): boolean</li> <li>size(): int</li> <li>toArray(): Object[]</li> <li>toString(): String</li> <li>height(t: RBNode&lt;T&gt;): int</li> <li>height(): int</li> <li>pathHeight(item: T): int</li> </ul>

org.eda1.actividad03.BS_AVL_RB_Tree.RBTree<T>.Treeliterator
<ul style="list-style-type: none"> <li>expectedModCount: int</li> <li>lastReturned: RBNode&lt;T&gt;</li> <li>nextNode: RBNode&lt;T&gt;</li> </ul>
<ul style="list-style-type: none"> <li>Treeliterator()</li> <li>hasNext(): boolean</li> <li>current(): T</li> <li>next(): T</li> <li>remove(): void</li> <li>checkIteratorState(): void</li> </ul>

org.eda1.actividad03.BS_AVL_RB_Tree.RBTree<T>.RBNode<T>
<ul style="list-style-type: none"> <li>BLACK: int</li> <li>RED: int</li> <li>parent: RBNode&lt;T&gt;</li> <li>left: RBNode&lt;T&gt;</li> <li>right: RBNode&lt;T&gt;</li> <li>color: int</li> <li>nodeValue: T</li> </ul>
RBNode(item: T, left: RBNode<T>, right: RBNode<T>, parent: RBNode<T>, color: int): void