Adicionalmente, se ha proporcionado todo lo relativo a la implementación del AVLTree, por lo que se pide que se EXPLIQUE en un documento (memoria) toda implementación suministrada, prestando especial atención a funciones principales de dicha estructura de datos como: add, remove, clear, contains, isEmpty, y al iterador iterator del tipo TreeIterator. También se pide, en dicha memoria, que se enumeren las ventajas e inconvenientes de la resolución de este problema mediante el uso de estructuras arbóreas (AVLTree) en lugar de colecciones lineales (ArrayList) como se hizo la práctica 1 (Ejercicio 02). Razone y justifique adecuadamente cada una de las ventajas e inconvenientes enumerados en el archivo PDF con nombre practica02_ejercicio02_AVLTree, almacenándose en la carpeta Memorias.

add: Añade un elemento igual que en el BSTree la diferencia es que equilibra el árbol al añadirlo.

remove: Elimina el elemento, comprueba el arbol y hace rotaciones si es necesario para equilibrarlo.

clear: Vacía el árbol poniendo el contador a 0 y el nodo root apunta a nulo.

contains: Recorre el árbol y devuelve un true si se encuentra dicho elemento.

isEmpty: Comprueba si el árbol esta nudo.

iterator: devuelve un iterador para recorrer el árbol.

TreeIterator: Clase que implementa la interfaz iterator para recorrer el árbol. El método iterator instancia un objeto de esta clase.
Para resolver este problema es preferible la implementación con una estructura como AVLTree ya que las consultas son mucho más eficientes en arboles binarios de búsqueda y pese a que la inserción y eliminación sea algo más costosas que en ArrayList, la relación sigue siendo ventajoso el AVLTree para este problema.