Java: Interfaces Set y Dictionary. Implementaciones AVLSet y AVLDictionary

La interfaz Set en Java

```
package dataStructures.set;

public interface Set<T> extends Iterable<T> {
        boolean isEmpty();

        int size();

        /**
         * Inserts new element in set. If element was already included, set is not modified
         * (this is not considered an error and thus no exception is thrown).
         * @param x Element to insert.
         */
        void insert(T x);

        boolean isElem(T x);

        /**
         * Removes element from set. If element is not in set, set is not modified
         * (this is not considered an error and thus no exception is thrown).
         * @param x Element to remove.
         */
        void delete(T x);
}
```

La implementación AVLSet en Java

```
package dataStructures.set;

import java.util.Iterator;

import dataStructures.searchTree.AVL;
import dataStructures.searchTree.SearchTree;

public class AVLSet<T extends Comparable<? super T>> implements Set<T>{

        private static class Nothing{};
        private Nothing nothing = new Nothing();

        private SearchTree<T,Nothing> tree;
        public AVLSet() {
                tree = new AVL<T,Nothing>();
        }

        public void delete(T elem) {
                tree.delete(elem);
        }

        public void insert(T elem) {
                tree.insert(elem,nothing);
        }

        public boolean isElem(T elem) {
                return tree.isElem(elem);
        }

        ...
```

La implementación AVLSet en Java

```
        ...

        public int size() {
                return tree.size();
        }

        public boolean isEmpty() {
                return tree.isEmpty();
        }

        public Iterator<T> iterator() {
                return tree.inOrder().iterator();
        }

        public String toString() {
                 String className =
                          getClass().getName().substring(getClass().
                          getPackage().getName().length()+1);
                String s = className+"(";
                Iterator<T> it = this.iterator();
                while(it.hasNext())
                          s += it.next() + (it.hasNext() ? "," : "");
                s += ")";
                return s;
        }
}
```

La interfaz Dictionary en Java

```
package dataStructures.dictionary;

import dataStructures.tuple.Tuple2;

public interface Dictionary<K, V> {
          boolean isEmpty();
          int size();

          /**
           * Inserts a new key/value association in dictionary. If key was already present
           * in dictionary, old value is replaced by {@code v} (different associations for same key
           * are not supported).
           * @param k Key in association.
           * @param v Value associated to key.
           */
          void insert(K k, V v);


          /**
           * Retrieves value associated to key {@code k}. If key is not
           * in dictionary, {@code null} is returned.
           * @param k Key for which associated value is sought.
           * @return Value associated to key or {@code null} if key is not in dictionary.
           */
          V valueOf(K k);

          /**
           * Tests whether an association with key {@code k} is included in dictionary.
           * @param k Key of association.
           * @return {@code true} if dictionary includes an association for key {@code k}, else {@code false}.
           */
          boolean isDefinedAt(K k);

          ...
```

La interfaz Dictionary en Java

```
    ...
    /**
     * Removes a key/value association from dictionary. If association with key {@code k} is not
     * in dictionary, dictionary is not modified (this is not considered an
     * error and thus no exception is thrown).
     * @param k Key of association to remove.
     */
    void delete(K k);

    /**
     * Retrieves an {@code Iterable} over all keys in dictionary.
     * Note that {@code remove} method is not supported in corresponding {@code iterator}.
     * Note also that dictionary structure or keys should not be modified during iteration as
     * iterator state may become inconsistent.
     * @see java.lang.Iterable
     * @return An {@code Iterable} over all keys in dictionary.
     */
    Iterable<K> keys();

    /**
     * Retrieves an {@code Iterable} over all values in dictionary.
     * Note that {@code remove} method is not supported in corresponding {@code iterator}.
     * Note also that dictionary structure or keys should not be modified during iteration as
     * iterator state may become inconsistent.
     * @see java.lang.Iterable
     * @return An {@code Iterable} over all keys in dictionary.
     */
    Iterable<V> values();

    /**
     * Retrieves an {@code Iterable} over all keys and values in dictionary.
     * Note that {@code remove} method is not supported in corresponding {@code iterator}.
     * Note also that dictionary structure or keys should not be modified during iteration as
     * iterator state may become inconsistent.
     * @see java.lang.Iterable
     * @return An {@code Iterable} over all keys in dictionary.
     */
    Iterable<Tuple2<K,V>> keysValues();
}
```

La implementación AVLDictionary en Java

```
package dataStructures.dictionary;
import dataStructures.searchTree.AVL;
import dataStructures.searchTree.SearchTree;
import dataStructures.tuple.Tuple2;


public class AVLDictionary<K extends Comparable<? super K>,V> implements Dictionary<K,V> {
          private SearchTree<K,V> tree;

          public AVLDictionary() {
                    tree = new AVL<K,V>();
          }

          public boolean isEmpty() {
                    return tree.isEmpty();
          }

          public int size() {
                    return tree.size();
          }

          public void insert(K k, V v) {
                    tree.insert(k, v);
          }

          public void delete(K k) {
                    tree.delete(k);
          }

          ...
```

La implementación AVLDictionary en Java

```
...
public V valueOf(K k) {
        return tree.search(k);
}

public boolean isDefinedAt(K k) {
        return tree.isElem(k);
}

public Iterable<K> keys() {
        return tree.inOrder();
}

public Iterable<V> values() {
        return tree.values();
}

public Iterable<Tuple2<K,V>> keysValues() {
        return tree.keysValues();
}

public String toString() {
        String className = getClass().getName().substring(getClass().
                                getPackage().getName().length()+1);
        String s = className+"(";
        if(!tree.isEmpty()) {
                for(Tuple2<K,V> t : tree.keysValues())
                        s += t._1()+"->"+t._2()+",";
                s = s.substring(0, s.length()-1);
        }
        s += ")";
  return s;
}
}
```