/\*

CS310, Summer 16

Alexander Giang

masc2169

\*/

package data\_structures;

import java.util.\*;

import java.util.Iterator;

import java.util.NoSuchElementException;

public class RedBlackTree<K,V> implements DictionaryADT<K,V> {

private TreeMap<K,V> tree;

public RedBlackTree() {

tree = new TreeMap<K,V>();

}

// Returns true if the dictionary has an object identified by

// key in it, otherwise false.

public boolean contains(K key) {

return tree.containsKey(key);

}

// Adds the given key/value pair to the dictionary. Returns

// false if the dictionary is full, or if the key is a duplicate.

// Returns true if addition succeeded.

public boolean add(K key, V value) {

if (tree.containsKey(key))

return false;

tree.put(key,value);

return true;

}

// Deletes the key/value pair identified by the key parameter.

// Returns true if the key/value pair was found and removed,

// otherwise false.

public boolean delete(K key) {

return tree.remove(key) != null;

}

// Returns the value associated with the parameter key. Returns

// null if the key is not found or the dictionary is empty.

public V getValue(K key) {

return tree.get(key);

}

// Returns the key associated with the parameter value. Returns

// null if the value is not found in the dictionary. If more

// than one key exists that matches the given value, returns the

// first one found.

public K getKey(V value){

Iterator<K> keyIter = keys();

Iterator<V> valueIter = values();

while(keyIter.hasNext()) {

K tmpK = keyIter.next();

V tmpV = valueIter.next();

if(((Comparable<V>) value).compareTo(tmpV) == 0)

return tmpK;

}

return null;

}

// Returns the number of key/value pairs currently stored

// in the dictionary

public int size() {

return tree.size();

}

// Returns true if the dictionary is at max capacity

public boolean isFull() {

return false;

}

// Returns true if the dictionary is empty

public boolean isEmpty() {

return tree.size() == 0;

}

// Returns the Dictionary object to an empty state.

public void clear() {

tree.clear();

}

// Returns an Iterator of the keys in the dictionary, in ascending

// sorted order. The iterator must be fail-fast.

public Iterator<K> keys() {

return tree.keySet().iterator();

}

// Returns an Iterator of the values in the dictionary. The

// order of the values must match the order of the keys.

// The iterator must be fail-fast.

public Iterator<V> values() { return null; }

}