ST.java

Below is the syntax highlighted version of ST.java from § Code. Here is the Javadoc.

```
/*************************************************************************
 *  Compilation:  javac ST.java
 *  Execution:    java ST
 *
 *  Sorted symbol table implementation using a java.util.TreeMap.
 *  Does not allow duplicates.
 *
 *  % java ST
 *
 *************************************************************************/

import java.util.Iterator;
import java.util.NoSuchElementException;
import java.util.SortedMap;
import java.util.TreeMap;

/**
 *  The <tt>ST</tt> class represents an ordered symbol table of generic
 *  key-value pairs.
 *  It supports the usual <em>put</em>, <em>get</em>, <em>contains</em>,
 *  <em>delete</em>, <em>size</em>, and <em>is-empty</em> methods.
 *  It also provides ordered methods for finding the <em>minimum</em>,
 *  <em>maximum</em>, <em>floor</em>, and <em>ceiling</em>.
 *  It also provides a <em>keys</em> method for iterating over all of the keys.
 *  A symbol table implements the <em>associative array</em> abstraction:
 *  when associating a value with a key that is already in the symbol table,
 *  the convention is to replace the old value with the new value.
 *  Unlike {@link java.util.Map}, this class uses the convention that
 *  values cannot be <tt>null</tt>&mdash;setting the
 *  value associated with a key to <tt>null</tt> is equivalent to deleting the key
 *  from the symbol table.
 *  <p>
```

```
 *  This implementation uses a balanced binary search tree. It requires that
 *  the key type implements the <tt>Comparable</tt> interface and calls the
 *  <tt>compareTo()</tt> and method to compare two keys. It does not call either
 *  <tt>equals()</tt> or <tt>hashCode()</tt>.
 *  The <em>put</em>, <em>contains</em>, <em>remove</em>, <em>minimum</em>,
 *  <em>maximum</em>, <em>ceiling</em>, and <em>floor</em> operations each take
 *  logarithmic time in the worst case.
 *  The <em>size</em>, and <em>is-empty</em> operations take constant time.
 *  Construction takes constant time.
 *  <p>
 *  For additional documentation, see <a href="http://introcs.cs.princeton.edu/44st">Section 4.4</a> of
 *  <i>Introduction to Programming in Java: An Interdisciplinary Approach</i> by Robert Sedgewick and Kevin Wayne.
 */
public class ST<Key extends Comparable<Key>, Value> implements Iterable<Key> {

   private TreeMap<Key, Value> st;

   /**
    * Initializes an empty symbol table.
    */
   public ST() {
       st = new TreeMap<Key, Value>();
   }


   /**
    * Returns the value associated with the given key.
    * @param key the key
    * @return the value associated with the given key if the key is in the symbol table
    *     and <tt>null</tt> if the key is not in the symbol table
    * @throws NullPointerException if <tt>key</tt> is <tt>null</tt>
    */
   public Value get(Key key) {
       if (key == null) throw new NullPointerException("called get() with null key");
       return st.get(key);
   }

   /**
    * Inserts the key-value pair into the symbol table, overwriting the old value
    * with the new value if the key is already in the symbol table.
    * If the value is <tt>null</tt>, this effectively deletes the key from the symbol table.
    * @param key the key
    * @param val the value
```

```
     * @throws NullPointerException if <tt>key</tt> is <tt>null</tt>
     */
    public void put(Key key, Value val) {
        if (key == null) throw new NullPointerException("called put() with null key");
        if (val == null) st.remove(key);
        else             st.put(key, val);
    }

    /**
     * Removes the key and associated value from the symbol table
     * (if the key is in the symbol table).
     * @param key the key
     * @throws NullPointerException if <tt>key</tt> is <tt>null</tt>
     */
    public void delete(Key key) {
        if (key == null) throw new NullPointerException("called delete() with null key");
        st.remove(key);
    }

    /**
     * Does this symbol table contain the given key?
     * @param key the key
     * @return <tt>true</tt> if this symbol table contains <tt>key</tt> and
     *     <tt>false</tt> otherwise
     * @throws NullPointerException if <tt>key</tt> is <tt>null</tt>
     */
    public boolean contains(Key key) {
        if (key == null) throw new NullPointerException("called contains() with null key");
        return st.containsKey(key);
    }

    /**
     * Returns the number of key-value pairs in this symbol table.
     * @return the number of key-value pairs in this symbol table
     */
    public int size() {
        return st.size();
    }

    /**
     * Is this symbol table empty?
     * @return <tt>true</tt> if this symbol table is empty and <tt>false</tt> otherwise
     */
```

```
public boolean isEmpty() {
    return size() == 0;
}

/**
 * Returns all keys in the symbol table as an <tt>Iterable</tt>.
 * To iterate over all of the keys in the symbol table named <tt>st</tt>,
 * use the foreach notation: <tt>for (Key key : st.keys())</tt>.
 * @return all keys in the sybol table as an <tt>Iterable</tt>
 */
public Iterable<Key> keys() {
    return st.keySet();
}

/**
 * Returns all of the keys in the symbol table as an iterator.
 * To iterate over all of the keys in a symbol table named <tt>st</tt>, use the
 * foreach notation: <tt>for (Key key : st)</tt>.
 * @return an iterator to all of the keys in the symbol table
 */
public Iterator<Key> iterator() {
    return st.keySet().iterator();
}

/**
 * Returns the smallest key in the symbol table.
 * @return the smallest key in the symbol table
 * @throws NoSuchElementException if the symbol table is empty
 */
public Key min() {
    if (isEmpty()) throw new NoSuchElementException("called min() with empty symbol table");
    return st.firstKey();
}

/**
 * Returns the largest key in the symbol table.
 * @return the largest key in the symbol table
 * @throws NoSuchElementException if the symbol table is empty
 */
public Key max() {
    if (isEmpty()) throw new NoSuchElementException("called max() with empty symbol table");
    return st.lastKey();
}
```

```java
    /**
     * Returns the smallest key in the symbol table greater than or equal to <tt>key</tt>.
     * @return the smallest key in the symbol table greater than or equal to <tt>key</tt>
     * @param key the key
     * @throws NoSuchElementException if the symbol table is empty
     * @throws NullPointerException if <tt>key</tt> is <tt>null</tt>
     */
    public Key ceil(Key key) {
        if (key == null) throw new NullPointerException("called ceil() with null key");
        SortedMap<Key, Value> tail = st.tailMap(key);
        if (tail.isEmpty()) throw new NoSuchElementException();
        return tail.firstKey();
    }

    /**
     * Returns the largest key in the symbol table less than or equal to <tt>key</tt>.
     * @return the largest key in the symbol table less than or equal to <tt>key</tt>
     * @param key the key
     * @throws NoSuchElementException if the symbol table is empty
     * @throws NullPointerException if <tt>key</tt> is <tt>null</tt>
     */
    public Key floor(Key key) {
        if (key == null) throw new NullPointerException("called floor() with null key");
        // headMap does not include key if present (!)
        if (st.containsKey(key)) return key;
        SortedMap<Key, Value> head = st.headMap(key);
        if (head.isEmpty()) throw new NoSuchElementException();
        return head.lastKey();
    }

    /**
     * Unit tests the <tt>ST</tt> data type.
     */
    public static void main(String[] args) {
        ST<String, String> st = new ST<String, String>();

       // insert some key-value pairs
        st.put("www.cs.princeton.edu",   "128.112.136.11");
        st.put("www.cs.princeton.edu",   "128.112.136.35");    // overwrite old value
        st.put("www.princeton.edu",      "128.112.130.211");
        st.put("www.math.princeton.edu", "128.112.18.11");
        st.put("www.yale.edu",           "130.132.51.8");
```

```
        st.put("www.amazon.com",         "207.171.163.90");
        st.put("www.simpsons.com",       "209.123.16.34");
        st.put("www.stanford.edu",       "171.67.16.120");
        st.put("www.google.com",         "64.233.161.99");
        st.put("www.ibm.com",            "129.42.16.99");
        st.put("www.apple.com",          "17.254.0.91");
        st.put("www.slashdot.com",       "66.35.250.150");
        st.put("www.whitehouse.gov",     "204.153.49.136");
        st.put("www.espn.com",           "199.181.132.250");
        st.put("www.snopes.com",         "66.165.133.65");
        st.put("www.movies.com",         "199.181.132.250");
        st.put("www.cnn.com",            "64.236.16.20");
        st.put("www.iitb.ac.in",         "202.68.145.210");


        StdOut.println(st.get("www.cs.princeton.edu"));
        StdOut.println(st.get("www.harvardsucks.com"));
        StdOut.println(st.get("www.simpsons.com"));
        StdOut.println();

        StdOut.println("ceil(www.simpsonr.com) = " + st.ceil("www.simpsonr.com"));
        StdOut.println("ceil(www.simpsons.com) = " + st.ceil("www.simpsons.com"));
        StdOut.println("ceil(www.simpsont.com) = " + st.ceil("www.simpsont.com"));
        StdOut.println("floor(www.simpsonr.com) = " + st.floor("www.simpsonr.com"));
        StdOut.println("floor(www.simpsons.com) = " + st.floor("www.simpsons.com"));
        StdOut.println("floor(www.simpsont.com) = " + st.floor("www.simpsont.com"));

        StdOut.println();

        StdOut.println("min key: " + st.min());
        StdOut.println("max key: " + st.max());
        StdOut.println("size:    " + st.size());
        StdOut.println();

        // print out all key-value pairs in lexicographic order
        for (String s : st.keys())
            StdOut.println(s + " " + st.get(s));
    }

}
```

Last updated: Sun Oct 6 13:54:46 EDT 2013.