Merge.java

Below is the syntax highlighted version of Merge.java from §2.2 Mergesort.

```
/******************************************************************************
 *  Compilation:  javac Merge.java
 *  Execution:    java Merge < input.txt
 *  Dependencies: StdOut.java StdIn.java
 *  Data files:   http://algs4.cs.princeton.edu/22mergesort/tiny.txt
 *                http://algs4.cs.princeton.edu/22mergesort/words3.txt
 *
 *  Sorts a sequence of strings from standard input using mergesort.
 *
 *  % more tiny.txt
 *  S O R T E X A M P L E
 *
 *  % java Merge < tiny.txt
 *  A E E L M O P R S T X                 [ one string per line ]
 *
 *  % more words3.txt
 *  bed bug dad yes zoo ... all bad yet
 *
 *  % java Merge < words3.txt
 *  all bad bed bug dad ... yes yet zoo    [ one string per line ]
 *
 ******************************************************************************/

/**
 *  The {@code Merge} class provides static methods for sorting an
 *  array using mergesort.
 *  <p>
 *  For additional documentation, see <a href="http://algs4.cs.princeton.edu/22mergesort">Section 2.2</a> of
 *  <i>Algorithms, 4th Edition</i> by Robert Sedgewick and Kevin Wayne.
 *  For an optimized version, see {@link MergeX}.
 *
 *  @author Robert Sedgewick
 *  @author Kevin Wayne
 */
public class Merge {

    // This class should not be instantiated.
    private Merge() { }

    // stably merge a[lo .. mid] with a[mid+1 ..hi] using aux[lo .. hi]
    private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi) {
        // precondition: a[lo .. mid] and a[mid+1 .. hi] are sorted subarrays
        assert isSorted(a, lo, mid);
        assert isSorted(a, mid+1, hi);

        // copy to aux[]
        for (int k = lo; k <= hi; k++) {
            aux[k] = a[k];
        }

        // merge back to a[]
        int i = lo, j = mid+1;
        for (int k = lo; k <= hi; k++) {
            if      (i > mid)              a[k] = aux[j++];
            else if (j > hi)               a[k] = aux[i++];
            else if (less(aux[j], aux[i])) a[k] = aux[j++];
            else                           a[k] = aux[i++];
        }
        // postcondition: a[lo .. hi] is sorted
        assert isSorted(a, lo, hi);
    }

    // mergesort a[lo..hi] using auxiliary array aux[lo..hi]
    private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
        if (hi <= lo) return;
        int mid = lo + (hi - lo) / 2;
        sort(a, aux, lo, mid);
        sort(a, aux, mid + 1, hi);
        merge(a, aux, lo, mid, hi);
    }

    /**
     * Rearranges the array in ascending order, using the natural order.
     * @param a the array to be sorted
     */
    public static void sort(Comparable[] a) {
        Comparable[] aux = new Comparable[a.length];
        sort(a, aux, 0, a.length-1);
        assert isSorted(a);
    }


   /***************************************************************************
    *  Helper sorting function.
    ***************************************************************************/

    // is v < w ?
    private static boolean less(Comparable v, Comparable w) {
        return v.compareTo(w) < 0;
    }

   /***************************************************************************
    *  Check if array is sorted - useful for debugging.
    ***************************************************************************/
    private static boolean isSorted(Comparable[] a) {
        return isSorted(a, 0, a.length - 1);
    }

    private static boolean isSorted(Comparable[] a, int lo, int hi) {
        for (int i = lo + 1; i <= hi; i++)
            if (less(a[i], a[i-1])) return false;
        return true;
    }


   /***************************************************************************
    *  Index mergesort.
    ***************************************************************************/
    // stably merge a[lo .. mid] with a[mid+1 .. hi] using aux[lo .. hi]
    private static void merge(Comparable[] a, int[] index, int[] aux, int lo, int mid, int hi) {

        // copy to aux[]
        for (int k = lo; k <= hi; k++) {
            aux[k] = index[k];
        }

        // merge back to a[]
        int i = lo, j = mid+1;
        for (int k = lo; k <= hi; k++) {
            if      (i > mid)                    index[k] = aux[j++];
            else if (j > hi)                     index[k] = aux[i++];
            else if (less(a[aux[j]], a[aux[i]])) index[k] = aux[j++];
            else                                 index[k] = aux[i++];
        }
    }
    /**
     * Returns a permutation that gives the elements in the array in ascending order.
     * @param a the array
     * @return a permutation {@code p[]} such that {@code a[p[0]]}, {@code a[p[1]]},
     *    ..., {@code a[p[N-1]]} are in ascending order
     */
    public static int[] indexSort(Comparable[] a) {
        int n = a.length;
        int[] index = new int[n];
        for (int i = 0; i < n; i++)
            index[i] = i;

        int[] aux = new int[n];
        sort(a, index, aux, 0, n-1);
        return index;
    }

    // mergesort a[lo..hi] using auxiliary array aux[lo..hi]
    private static void sort(Comparable[] a, int[] index, int[] aux, int lo, int hi) {
        if (hi <= lo) return;
        int mid = lo + (hi - lo) / 2;
        sort(a, index, aux, lo, mid);
        sort(a, index, aux, mid + 1, hi);
        merge(a, index, aux, lo, mid, hi);
    }

    // print array to standard output
    private static void show(Comparable[] a) {
        for (int i = 0; i < a.length; i++) {
            StdOut.println(a[i]);
        }
    }

    /**
     * Reads in a sequence of strings from standard input; mergesorts them;
     * and prints them to standard output in ascending order.
     *
     * @param args the command-line arguments
     */
    public static void main(String[] args) {
        String[] a = StdIn.readAllStrings();
        Merge.sort(a);
        show(a);
    }
}
```

Last updated: Tue Aug 30 10:09:18 EDT 2016.