

TestSorter Class

```
import java.util.Random;
/**
 * Class Sorter provides sorting methods on linear arrays of double.
 * Methods sortSelection(), sortInsertion(), sortBubble() and sortMerge()
 * sort the array that is passed as an argument to them.
 *
 * Class Sorter uses the private methods minLocationFrom() and swap() to
 * facilitate the development of the sorting methods. It provides the
 * public method merge() which merges two sorted arrays. It also provides a
random
 * number generator which fills an array and array printing methods.
 *
 * @author (Antonios Symvonis)
 * @version (11 March 2004)
 */

public class Sorter
{
    //static random number generator
    private static Random random=new Random(0);

    /**
     * Prints the elements of an array, all at the same line.
     *
     * @param a The array of which the ellements are printed.
     */
    public static void printH(double a[])
    {
        int len=a.length;
        System.out.print("{ ");
        for( int i=0; i< len-1; i++)
            System.out.print(a[i] + ", " );
        System.out.print(a[len-1] + " }");
    }//printH

    /**
     * Prints the elements of an array, each at a different line.
     *
     * @param a The array of which the ellements are printed.
     */
    public static void printV(double a[])
    {
        int len=a.length;
        for( int i=0; i< len; i++)
            System.out.println(a[i]);
    }//printV

    /**
     * Creates an exact copy of an array.
     *
     * @param a The array to be copied. It cannot be NULL.
     * @return An exact copy of the argument array.
     */
    public static double[] clone(double original[])
```

```

{
    double copy[] = new double[original.length];
    int len=original.length;
    for( int i=0; i<len; i++)
        copy[i]=original[i];

    return copy;
}//clone

/**
 * Fills a linear array with random data in the range [0..1). The random
 * numbers have precision of 2 decimal digits.
 *
 * @param a      The array to be filled. It cannot be NULL.
 */
public static void fillRandomData(double a[])
{
    int len=a.length;

    for (int i=0; i< len; i++)
        a[i]= (double) random.nextInt(100)/100;
}//fillRandomData

/**
 * Swaps the elements of the array at the specified positions
 *
 * @param a      The array of which the elements are swapped.
 * @param pos1   Position of the first element.
 * @param pos2   Position of the second element.
 */
private static void swap(double a[], int pos1, int pos2)
{
    double temp;
    temp=a[pos1];
    a[pos1]=a[pos2];
    a[pos2]=temp;
}//swap

/**
 * Starting from a specified position of an array, it identifies the
 * location of the maximum element of the array. The specified position is
 * included in the search.
 *
 * @param a      The array to be searched.
 * @param start  The search for the maximum starts from this position.
 * @return       The index of the maximum element.
 */
private static int maxLocationFrom(double a[], int start)
{
    int len=a.length;
    int maxLoc=start;

    for (int i=start; i<len; i++)
        if (a[maxLoc] <a[i])
            maxLoc=i;
    return maxLoc;
}//maxLocationFrom

/**

```

```

/*
 * Starting from a specified position of an array, it identifies the
 * location of the minimum element of the array. The specified position is
 * included in the search.
 *
 * @param a           The array to be searched.
 * @param start      The search for the minimum starts from this position.
 * @return           The index of the minimum element.
 */
private static int minLocationFrom(double a[], int start)
{
    int len=a.length;
    int minLoc=start;

    for (int i=start; i<len; i++)
        if (a[minLoc] >a[i])
            minLoc=i;
    return minLoc;
}//minLocationFrom

/**
 * Extracts a portion of an array.
 *
 * @param a           The array to be copied. It cannot be NULL.
 * @param start      The index of the first element to be extracted.
 * @param end        The index of the last element to be extracted.
 *                   It holds that (start <= end).
 * @return           The extracted array.
 */
public static double[] extract(double original[], int start, int end)
{
    int len=end-start+1;
    double copy[] = new double[len];
    for( int i=0; i<len; i++)
        copy[i]=original[start+i];

    return copy;
}//clone

/**
 * Sorts an array in O(n^2) time by using the "selection-sort" method.
 *
 * @param a           The array to be sorted. It cannot be NULL.
 */
public static void sortSelection(double a[])
{
    int len=a.length;
    int minLoc;
    for (int start=0; start < len-1; start++){
        minLoc=minLocationFrom(a,start);
        swap(a,start, minLoc);
    }
}//sortSelection

/**
 * Sorts an array in O(n^2) time by using the "insertion-sort" method.
 *
 * @param a           The array to be sorted. It cannot be NULL.
 */
public static void sortInsertion(double a[])

```

```

{
    int len=a.length;

    for (int i=1; i < len; i++){
        //Insert the i-th element into the sorted (from the previous
        iterations) segment a[0...i-1] of the array.

        double temp=a[i];
        int ip=0;           // will hold the location at which a[i] will be
        placed at the end of the iteration
        while ( (a[ip] <= temp) && (ip<i))
            ip++;
        //now ip contains the correct insertion point.

        //Shift the remaining elements in the segment 1 position to the right
        for (int j=i-1; j>=ip; j--)
            a[j+1]=a[j];

        a[ip]=temp;
    }
}//sortInsertion

/***
 * Sorts an array in O(n^2) time by using the "Bubble-sort" method.
 *
 * @param a      The array to be sorted. It cannot be NULL.
 */
public static void sortBubble(double a[])
{
    int len=a.length;
    int completed;
    for (completed=0; completed < len; completed++)
        for (int i=len-1; i>completed; i--)
            if (a[i] < a[i-1])
                swap(a,i,i-1);
}//sortBubble

//-----
/***
 * Merges two sorted arrays.
 *
 * @param a1      The first of the two arrays to be merged. It must be sorted
and cannot be NULL.
 * @param a2      The second of the two arrays to be merged. It must be
sorted and cannot be NULL.
 * @return         The merged array.
 */
public static double[] merge(double a1[], double a2[])
{
    double result[]=new double[a1.length+a2.length];
    int pos1=0;
    int pos2=0;
    int posResult=0;
    int len1=a1.length;
    int len2=a2.length;

    while( (pos1 < len1) && (pos2 < len2)){
        if (a1[pos1] < a2[pos2]){

```

```

        result[posResult]=a1[pos1];
        pos1++;
    }
    else {
        result[posResult]=a2[pos2];
        pos2++;
    }
    posResult++;
}
if (pos1==len1) //a1 was exhausted first
    while(pos2<len2){
        result[posResult]=a2[pos2];
        pos2++;
        posResult++;
    }
else
    while(pos1<len1){
        result[posResult]=a1[pos1];
        pos1++;
        posResult++;
    }
return result;
}//merge

```

/** * Sorts an array in $O(n\log n)$ time by using the "merge-sort" method. * This implementation is not in place, that is, it uses extra additional space. *

* @param a The array to be sorted. It cannot be NULL.

*/

```

public static void sortMerge(double a[])
{
    int len=a.length;
    if (len==1)
        return;
    else{
        //create an copy of the input and then sort it
        double aux[]=clone(a);
        int len1=len/2;
        double ar1[]=extract(aux, 0,len1-1);
        double ar2[]=extract(aux, len1,len-1);

        sortMerge(ar1);
        sortMerge(ar2);
        aux=merge(ar1,ar2);

        //copy the result back to the input array
        for (int i=0; i< len; i++)
            a[i]=aux[i];
    }
}

```

Sorter Class

```

/**
 * Used to test the static method of class Sorter.

```

```
  
*  
* @author (Antonios Symvonis)  
* @version (11 April 2002)  
*/  
public class TestSorter  
{  
    public static void main(String args[])  
    {  
  
        //arrays used in testing the methods of class Sorter  
        double data1[]=new double[10];  
        double data2[]=new double[10];  
        double data3[]=new double[10];  
        double data4[]=new double[10];  
  
        //test fillRandomData  
        Sorter.fillRandomData(data1);  
        Sorter.fillRandomData(data2);  
        Sorter.fillRandomData(data3);  
        Sorter.fillRandomData(data4);  
  
        //print test data  
        System.out.println("-----");  
        System.out.println("The test arrays are:\n");  
        System.out.print("data1[]:  ");  
        Sorter.printH(data1);  
        System.out.println();  
        System.out.print("data2[]:  ");  
        Sorter.printH(data2);  
        System.out.println();  
        System.out.print("data3[]:  ");  
        Sorter.printH(data3);  
        System.out.println();  
        System.out.print("data4[]:  ");  
        Sorter.printH(data4);  
        System.out.println("\n");  
  
        //test sortSelection  
        //    System.out.println("Testing sortSelection...");  
        //    System.out.println("Array \"data1[]\" before and after  
sortSelection:");  
        //    Sorter.printH(data1);  
        //    System.out.println(" (before sorting)");  
        //    Sorter.sortSelection(data1);  
        //    Sorter.printH(data1);  
        //    System.out.println(" (after sorting)\n");  
  
        //test sortInsertion  
        //    System.out.println("Testing sortInsertion...");  
        //    System.out.println("Array \"data2[]\" before and after  
sortSelection:");  
        //    Sorter.printH(data2);  
        //    System.out.println(" (before sorting)");  
        //    Sorter.sortInsertion(data2);  
        //    Sorter.printH(data2);  
        //    System.out.println(" (after sorting)\n");  
  
        //test sortBubble  
        //    System.out.println("Testing sortBubble...");  
        //    System.out.println("Array \"data3[]\" before and after sortBubble:");  
        //    Sorter.printH(data3);  
    }  
}
```

```
//      System.out.println("  (before sorting)");
//      Sorter.sortInsertion(data3);
//      Sorter.printH(data3);
//      System.out.println("  (after sorting)\n");

//test  sortMerge
//      System.out.println("Testing sortMerge...");
//      System.out.println("Array \"data4[]\" before and after sortMerge:");
//      Sorter.printH(data4);
//      System.out.println("  (before sorting)");
//      Sorter.sortInsertion(data4);
//      Sorter.printH(data4);
//      System.out.println("  (after sorting)\n");

System.out.println("\n-----END OF \"main\" (class TestSorter)-----\n");
}//main
}
```

