שלושת המיונים בצורה רקורסיבית, לפי גישת התכנות מונחה העצמים

המחלקה Sort שלהלן מביאה שוב את שלושת המיונים שהוצגו ביחידה 16 בצורה רקורסיבית.

במחלקה שדה אחד data_ שהוא המערך. שלושת המיונים מופיעים כשיטות לא סטטיות, אלא שיטות שפועלות על האובייקט, ולכן המערך לא מועבר כפרמטר, שלא כמו בהרצאה.

הוספנו שתי שיטות, אחת ()enterNumbers לקריאת נתונים מהקלט והכנסתם למערך, והשניה ()printArray להדפסת המערך על הפלט.

```
import java.util.Scanner;

/**
 * This class demonstrates three sorting algorithms of an array.
 *
 * @author (Tamar Vilner)
 *
*/

public class Sort
{
    // _data = the array
    private int [] _data;
    // MAX = the length of the array
    final int MAX = 6;

    /**
     * Constructor for objects of class Sorting
     */
    public Sort()
    {
         _data = new int [MAX];
    }

     /**
     * enterNumbers is a method that fills the array with data from
     * the user
     */
    public void enterNumbers()
    {
        Scanner scan = new Scanner (System.in);
        for (int i=0; i<_data.length; i++)
        {
            System.out.print("enter an integer ");
            _data[i] = scan.nextInt();
        }
    }

    /**
     * printArray prints the array content
     */
    public void printArray()
    {
        System.out.print("The array is: ");
        for (int i=0; i<_data.length; i++)
            System.out.print("\t"+ _data[i]);
        System.out.println();
    }
```

```
//-------------------------------------------------------------------//
// Selection Sort
//-------------------------------------------------------------------//

    /**
     * selectionSort sorts the array using the selelction-sort
     * algorithm
     * @param lo the lowest index of the array
     * @param hi the highest index of the array
     */
        public void selectionSort(int lo, int hi)
        {
            // data[0]…data[lo-1] contain the largest values in data,
            //  in descending order
            if (lo < hi)      //subarray has more than one element
            {
                swap( lo, findMaximum(lo, hi));
                selectionSort(lo+1, hi);
            }
        }

        private int findMaximum( int lo, int hi)
        {
           // finds the maximum in the array betwen the indexes
           // lo and hi
           if (lo == hi)  return lo;
           else
           {
               int locationOfMax = findMaximum( lo+1, hi);
               if (_data[lo] > _data[locationOfMax])  return lo;
               else  return locationOfMax;
           }
        }

        private void swap( int first, int second)
        {
                //swaps between the contents of the array cells in
                //indexes first and second
                   int  temp;
                   temp = _data[first];
                   _data[first] = _data[second];
                   _data[second] = temp;
        }
```

```
//------------------------------------------------------------------//
// Insertion Sort
//------------------------------------------------------------------//

    /**
     * insertionSort sorts the array using the insertion-sort
     * algorithm
     * @param hi the highest index of the array
     */
       public  void insertionSort( int hi)
       {
           //  Sort data[0]…data[hi]
           if (hi > 0)
           {
               insertionSort( hi-1);
               insertInOrder( hi, _data[hi]);
           }
       }

       private  void insertInOrder(int hi, int x)
       {
           // Insert x into data[0]…data[hi-1], filling
           // in data[hi] in the process.
           // data[0]…data[hi-1] are sorted.
           if ( (hi == 0) || (_data[hi-1] >= x) )
               _data[hi] = x;
           else
           {
               _data[hi] = _data[hi-1];
               insertInOrder(hi-1, x);
           }
       }
```

```
//------------------------------------------------------------------//
// Quick Sort
//------------------------------------------------------------------//

    /**
     * quickSort sorts the array using the quick-sort algorithm
     */

        public   void quicksort() {
                quicksort( 0, _data.length-1);
            }


        private  void quicksort(int lo, int hi)
        {
            int m;

            if (hi > lo+1)
            {       // there are at least 3 elements
                     // so sort recursively
                     m = partition(lo, hi);
                     quicksort( lo, m-1);
                     quicksort( m+1, hi);
            }
            else    // 0, 1, or 2 elements, so sort directly
                    if (hi == lo+1  &&  _data[lo] > _data[hi])
                        swap(lo, hi);
        }

        private  int medianLocation( int i, int j, int k)
        {
            if (_data[i] <= _data[j])
                if (_data[j] <= _data[k])
                        return j;
                else if (_data[i] <= _data[k])
                                return k;
                          else    return i;
            else    // _data[j] < _data[i]
                if (_data[i] <= _data[k])
                        return i;
                else if (_data[j] <= _data[k])
                                return k;
                           else    return j;
        }


        private  int partition(int lo, int hi)
        {
            // Choose middle element among a[lo]…a[hi],
            // and move other elements so that a[lo]…a[m-1]
            // are all less than a[m] and a[m+1]…a[hi] are
            // all greater than a[m]
            //
            // m is returned to the caller
            swap(a, lo, medianLocation(a, lo+1, hi, (lo+hi)/2));
            int m = partition( lo+1, hi, _data[lo]);
            swap( lo, m);
            return m;
        }
        private  int partition(int lo, int hi, int pivot)
        {
                if (hi == lo)
                    if (_data[lo] < pivot)
                                return lo;
                    else
                                return lo-1;
                else if (_data[lo] <= pivot) // a[lo] in correct half
                                return partition(lo+1, hi, pivot);
                          else
                          {               // a[lo] in wrong half
                                swap( lo, hi);
                                return partition( lo, hi-1, pivot);
                          }
         }


}  // end of class sorting
```