```java

import java.util.Arrays;


public class StaticArrayExercises {

    public static void main(String[] args) {

        // You do not need to handle the User Interface (UI) first.
        // Instead you can run the JUnit test cases found in StaticArrayTest.java

        // Construct and initialize an array of random integer values, then pass into the methods ...
        //double mean = calculateMean(/*  Pass array into method */  );
        //double median = calculateMedian(/*  Pass array into method */  );
        //double mode = calculateMode(/*  Pass array into method */  );

        // Keep going ...









    }

    /**
     * Calculates the mean of a given static integer array of positive values
     * @param values an array of positive integer values
     * @return the mean
     */
```

```
33      public static double calculateMean(int[] values)  {
34          double mean = 0;
35          if (values.length == 0) {
36              return mean;
37          }
38          int sum = 0;
39          for (int i = 0; i < values.length; i++)
40              sum =  sum + values[i];
41          mean = (double)sum/values.length;
42          return mean;
43
44
45
46      }
47
48      /**
49       * Calculates the median of a given static integer
    array of positive values
50       * @param values an array of positive integer values
51       * @return the mode
52       */
53      public static double calculateMedian(int[] values)  {
54          double median = 0;
55          //to look at array
56          // System.out.println(Arrays.toString(values));
57          //to sort array
58          Arrays.sort(values);
59          System.out.print(Arrays.toString(values));
60
61          if (values.length == 0) {
62              return median;
63          }
64
65          else if (values.length %2 == 0) {
66              int medianpos = (values.length)/2;
67              int secondmedianpos = ((values.length)/2) -
    1;
68              median = (values[medianpos] +
    values[secondmedianpos]) / 2.0;
```

```
 69            return median;
 70        }
 71
 72        else {
 73            int pos = ((values.length - 1)  / 2);
 74            median = values[pos];
 75            return median;
 76        }
 77    }
 78
 79    /**
 80     * Calculates the mode of a given static integer
   array of positive values
 81     * It is technically possible for a list of numbers
   to have multiple modes or no mode.
 82     * For this assignment you are not concerned with
   either of these cases.
 83     * @param values an array of positive integer values
 84     * @return the mode
 85     */
 86    public static int calculateMode(int[] values)  {
 87        Arrays.sort(values);
 88        int i = 0;
 89        int x = 1;
 90        int realmode = values[0];
 91        int storedmode = 1;
 92
 93        while (i < (values.length - 1)) {
 94            if (values[i] == values[i+1]) {
 95                x = x + 1;
 96            }
 97
 98            else {
 99                x = 1;
100            }
101
102            if (x > storedmode) {
103                    storedmode = x;
104                    realmode = values[i];
```

```
105                 }
106             i = i+1;
107 
108             }
109 
110     return realmode;
111 
112     }
113 
114 
115 
116 
117     /**
118      * Determine if the number that the user entered is in the array of values.
119      * @param values an array of integer values
120      * @param valToFind the integer to find
121      * @return true if valToFind is in array values; false otherwise
122      */
123     public static boolean linearSearch(int[] values, int valToFind)  {
124         boolean found = false; // Assume the value is not in the array
125         for (int i = 0; i< values.length; i++)
126             if (valToFind == values[i]) {
127             found = true;
128             }
129 
130         return found;
131     }
132 
133     /**
134      * Find the position of the first element that is larger than 30
135      * @param values an array of integer values
136      * @return the position (starting from 0) of the first element that is larger than 30, -1 if not found
137      */
```

```
138     public static int positionFind(int[] values)  {
139         int position = -1;  // Assume a value larger than
    30 is not in the array
140 
141         for (int i = 0; i< values.length; i++)
142             if (values[i]>30) {
143                 position = i;
144                 break;
145             }
146         return position;
147     }
148 
149 
150     /**
151      * A run is a sequence of adjacent repeated values.
152      * Write a program that generates a sequence of 20
    random die tosses and that prints the die values,
153      * marking the runs by including them in parentheses,
    like this:
154      * 1 2 (5 5) 3 1 2 4 3 (2 2 2 2) 3 6 (5 5) 6 3 1
155      * @param values an array with 20 random die tosses
    between 1 and 6, inclusive
156      */
157     public static String runs(int[] values)  {
158         String result = new String();
159         int x = 0;
160         if (values[0]==values[1]) {
161             result += "(";
162         }
163         result += values[0];
164 
165         for (x = 1; x < values.length-1; x++) {
166 
167             if (values[x]==values[x-1] && values[x] !=
    values[x+1]) {
168                 result += values[x];
169                 result += ")";
170             }
171 
```

```
172             else if (values[x]!=values[x-1] &&
    values[x]==values[x+1]) {
173                 result += "(";
174                 result += values[x];
175             }
176 
177             else {
178                 result += values[x];
179             }
180 
181         }
182 
183         if (values[values.length-1] ==
    values[values.length-2]) {
184             result += values[values.length-1];
185             result += ")";
186         }
187         else {
188             result += values[values.length-1];
189         }
190         return result;
191     }
192     /**
193      * An n x n matrix that is filled with the numbers 1,
    2, 3, … , n2 is
194      * a magic square if the sum of the elements in each
    row, in each column, and in the two diagonals is the same
    value
195      * @param n the size of the magic square where n is
    odd
196      * @return a magic square of size n-by-n where n is
    odd, or null otherwise
197      */
198     public static int[][] generateMagicSquare(int n) {
199 
200         if (n % 2 == 0) return null; // only odd n
201 
202         int[][] magic = new int[n][n];
203         int i = n - 1;
```

```
204         int j = (n - 1) / 2;
205
206         for (int x = 1; x <= n * n; x++) {
207         magic[i][j] = x;
208
209         int newi = (i + 1) % n;
210             int newj = (j + 1) % n;
211
212             if (magic[newi][newj] != 0) {
213                 newi = (i - 1 + n) % n;
214                 newj = j;
215             }
216
217             i = newi;
218             j = newj;
219         }
220
221         return magic;
222     }
223}
```