```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
     */
    public static double calculateMean(int[] values)  {
        double mean = 0;

        int sum =0;
        if (values.length == 0) {
            mean=0;
        } else {


        for (int i=0; i<values.length; i++) {
        sum = sum + values[i];
        }

         mean = (double)sum/(values.length); }

         return mean;    }
```

```
40
41      /**
42       * Calculates the median of a given static integer array of
    positive values
43       * @param values an array of positive integer values
44       * @return the mode
45       */
46      public static double calculateMedian(int[] values)  {
47          double median = 0;
48
49
50          Arrays.sort(values);
51
52          if (values.length%2== 0) {
53          int med = values[(values.length/2)-1];
54          int medi= values[(values.length/2)];
55           median = (med + medi)/(double)2;
56          } else {
57              median = (int) values[(values.length/2)];
58          }
59
60          return median;
61      }
62      /**
63       * Calculates the mode of a given static integer array of
    positive values
64       * It is technically possible for a list of numbers to have
    multiple modes or no mode.
65       * For this assignment you are not concerned with either of
    these cases.
66       * @param values an array of positive integer values
67       * @return the mode
68       */
69      public static int calculateMode(int[] values)  {
70      int mode = -1;
71      int maxTimes = -1;
72      for ( int i=0; i<values.length; i++) {
73          int modecount = 0;
74          for (int j=0; j<values.length; j++) {
75              if (values[i]==values[j]) {
76                  modecount++;
77              }
78          }
79          if (modecount > maxTimes) {
80              mode = values[i];
```

```
 81              maxTimes = modecount;
 82          }
 83      }
 84      return mode;
 85
 86          }
 87
 88
 89
 90      /**
 91       * Determine if the number that the user entered is in the
    array of values.
 92       * @param values an array of integer values
 93       * @param valToFind the integer to find
 94       * @return true if valToFind is in array values; false
    otherwise
 95       */
 96      public static boolean linearSearch(int[] values, int valToFind)
    {
 97          boolean found = false;  // Assume the value is not in the
    array
 98
 99          for (int i=0; i<values.length; i++) {
100           if (valToFind == values[i]) {
101               found = true;
102           }
103
104           }
105          return found;
106      }
107
108      /**
109       * Find the position of the first element that is larger than
    30
110       * @param values an array of integer values
111       * @return the position (starting from 0) of the first element
    that is larger than 30, -1 if not found
112       */
113      public static int positionFind(int[] values)  {
114          int position = -1;  // Assume a value larger than 30 is not
    in the array
115
116
117          for (int i=0; i<values.length; i++) {
118
```

```
119
120             if (values[i]>30) {
121                 position = i;
122                 return i;
123             }
124         }
125
126         return position;
127     }
128
129
130     /**
131      * A run is a sequence of adjacent repeated values.
132      * Write a program that generates a sequence of 20 random die tosses and that prints the die values,
133      * marking the runs by including them in parentheses, like this:
134      * 1 2 (5 5) 3 1 2 4 3 (2 2 2 2) 3 6 (5 5) 6 3 1
135      * @param values an array with 20 random die tosses between 1 and 6, inclusive
136      */
137     public static String runs(int[] values)  {
138         String result = new String(); // Start with an empty String as the result and "add/concatenate" to it with +
139
140         if (values[0] == values[1]) {
141             result = result + "(" + values[0];
142         } else {
143             result = result + values[0];
144         }
145
146     for (int i=1; i<values.length-1; i++) {
147         if (values[i+1]== values[i] && values[i-1]!= values[i]) {
148             result = result + " "+ "(" + values[i];
149         } else if (values[i+1]!= values[i] && values[i-1]== values[i]) {
150             result = result + " " + values[i] +")";
151         } else
152             result = result + " " + values[i];
153     }
154
155
156     if (values[values.length-2] == values[values.length-1]) {
157         result = result + " " + values[values.length-1] + ")";
158     } else {
```

```
159         result = result + " " + values[4];
160     }
161         return result;
162 }
163 
164     /**
165      * An n x n matrix that is filled with the numbers 1, 2, 3, … , n2 is
166      * a magic square if the sum of the elements in each row, in each column, and in the two diagonals is the same value
167      * @param n the size of the magic square where n is odd
168      * @return a magic square of size n-by-n where n is odd, or null otherwise
169      * 2-d arrays -- int [][] matrix = new int [5][5] first is row and second is columns
170      * 1-25 numbers cause n=5
171      * wrap back up to top after reaching bottom
172      * if spot has been filled, move up
173      */
174     public static int[][] generateMagicSquare(int n)   {
175 
176         if (n % 2 == 0)  // Return null if n is even (this is a different algorithm)
177             return null;
178 
179         int[][] magic = new int[n][n];  // Construct an n-by-n array where n is odd
180         int rows = n/2;
181         int columns = n-1;
182 
183         magic [columns][rows] =1;
184 
185     for (int i=2; i<= Math.pow(n,2); i++) {
186         if (magic[(columns+1)%n] [(rows+1)%n] == 0) {
187         columns = (columns +1)%n;
188         rows = (rows+1)%n;
189 
190             magic [columns][rows]= i;
191 
192     } else {
193         columns = columns-1;
194         magic [columns][rows] = i;
195     }
196 }
197 
```

```
198         return magic; }
199 }
200
```