```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 inStaticArrayTest.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;
        for (int i=0;i<values.length ; i++) {
                mean+=values[i];
        }
        if (mean != 0) {
                mean /= values.length;
        }
        return mean;
}
/**
* Calculates the median of a given static integer array of positive
values
* @param values an array of positive integer values
* @return the mode
*/
public static double calculateMedian(int[] values) {
        double median = 0;
        Arrays.sort(values);
        if(values.length%2==0) {
                median = (values[values.length/2] + values[values.length/2-1])/2.0;
        }else {
                median = values[values.length/2];
        }
        return median;
}
/**
* Calculates the mode of a given static integer array of positive values
* It is technically possible for a list of numbers to have multiple modes
or no mode.
* For this assignment you are not concerned with either of these
cases.
* @param values an array of positive integer values
* @return the mode
*/
public static int calculateMode(int[] values) {
    int mode = -1;
    int count = 1;
    int j = 1;
    int check = 0;
    Arrays.sort(values);
    for (int i=0; i<values.length; i++) {
        while((i+j)<values.length && values[i] == values[i+j]) {
            count++;
            j++;
        }
        if(check<count) {
            check = count;
            mode = values[i];
        }
        count = 1;
        j=1;
    }
    return mode;
}
/**
* Determine if the number that the user entered is in the array of
values.
* @param values an array of integer values
* @param valToFind the integer to find
* @return true if valToFind is in array values; false otherwise
*/
public static boolean linearSearch(int[] values, int valToFind) {
    boolean found = false; // Assume the value is not in the array
    for (int i = 0; i<values.length; i++) {
        if(values[i]==valToFind) {
            found = true;
        }
    }
    return found;
}
/**
* Find the position of the first element that is larger than 30
* @param values an array of integer values
* @return the position (starting from 0) of the first element that is
larger than 30, -1 if not found
*/
public static int positionFind(int[] values) {
    int position = -1; // Assume a value larger than 30 is not in the array
    for (int i=0; i<values.length; i++) {
        if(values[i]>30) {
            position = i;
```

```
                return position;
            }
        }
        return position;
}
/**
* A run is a sequence of adjacent repeated values.
* Write a program that generates a sequence of 20 random die tosses
and that prints the die values,
* marking the runs by including them in parentheses, like this:
* 1 2 (5 5) 3 1 2 4 3 (2 2 2 2) 3 6 (5 5) 6 3 1
* @param values an array with 20 random die tosses between 1 and
6, inclusive
*/
public static String runs(int[] values) {
        String result = new String(); // Start with an empty String as the result and "add/concatenate" to it
with +
        int j=1;
        int count=1;
        for (int i=0; i<values.length; i+=count) {
            if (i+j<values.length && values[i]==values[i+j]) {
                if (i==0) {
                    result+= "("+values[i];
                }else {
                result+=" ("+values[i];
                }
                while(i+j<values.length && values[i]==values[i+j]) {
                    result+=" "+values[i+j];
                    j++;
                }
                count = j;
                result+=")";
                j=1;
            }else {
                if (i==0) {
                    result+=values[i];
                }else {
                result+=" " + values[i];
                }
                count = 1;
            }
        }
        return result;
}
/**
* An n x n matrix that is filled with the numbers 1, 2, 3, , n2 is
* a magic square if the sum of the elements in each row, in each
column, and in the two diagonals is the same value
* @param n the size of the magic square where n is odd
* @return a magic square of size n-by-n where n is odd, or null
otherwise
*/
public static int[][] generateMagicSquare(int n) {
        if (n % 2 == 0) // Return null if n is even (this is a different algorithm)
        return null;
        int[][] magic = new int[n][n]; // Construct an n-by-n array where n is odd
        int y = n-1;
        int x = n/2;
        magic[y][x]=1;
        for(int i=2; i<=Math.pow(n,2); i++) {
            if(magic[(y+1)%n][(x+1)%n]==0) {
                y=(y+1)%n;
                x=(x+1)%n;
                magic[y][x]=i;
            }else {
                y--;
                magic[y][x]=i;
            }
        }
        return magic;
}
}
```