```
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 ...
		System.out.println(Arrays.deepToString(generateMagicSquare(3)));
	}

	/**
	 * 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;
		double sum = 0;
		for (int i = 0; i < values.length; i++) {
			sum += values[i];
		}
		mean = sum / values.length;
		if (values.length == 0) {
			mean = 0;
		}
		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 ﷺmultiple modesﷺ or ﷺno
 * modeﷺ. 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 cMode = 0;
	int cFrequency = 0;
	int fFrequency = 0;
	Arrays.sort(values);
	for (int i = 0; i < values.length; i++) {
		if (cMode == values[i]) {
			cFrequency++;
		} else {
			cMode = values[i];
			cFrequency = 1;
		}
		if (cFrequency > fFrequency) {
			mode = cMode;
			fFrequency = cFrequency;
		}
	}
	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 (valToFind == values[i]) {
				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; position == -1 && i < values.length; i++) {
			if (values[i] > 30) {
				position = i;
			}
		}
		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 appeared = 0;
		if (values.length > 1 && values[0] == values[1]) {
			result += "(" + values[0];
```

```
                appeared++;
        } else {
                result += values[0];
        }

        for (int i = 1; i < values.length - 1; i++) {
                if (values[i] == values[i + 1] && appeared == 0) {
                        result += " (" + values[i];
                        appeared++;
                } else if ((appeared > 0 && values[i] == values[i + 1]) || appeared == 0) {
                        result += " " + values[i];
                } else if (appeared > 0 && values[i] != values[i + 1]) {
                        appeared = 0;
                        result += " " + values[i] + ")";
                }
        }
        if (values.length > 1 && values[values.length - 1] == values[values.length - 2]) {
                result += " " + values[values.length - 1] + ")";
        } else {
                result += " " + values[values.length - 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 number = 1;
        int[][] magic = new int[n][n]; // Construct an n-by-n array where n is odd
        int row = n-1;
        int column = n/2;

        while(number <= n*n) {
```

```
                magic[row][column] = number;
                number++;

                if(magic[(row+1)%n][(column+1)%n] !=0) {
                        row = (row-1)%n;
                } else {
                        row= (row+1)%n;
                        column = (column+1)%n;
                }
        }


        return magic;
    }

}
```