JMAAB Ch. 12, #24 - 28

24. `i - j == 0 || i + j == n;`

25. A method that returns the value of the largest positive element in a 2-D array, or 0 if all its elements are negative:

```
/**
 * Method to find the greatest positive
 * value within the given array.
 * @return the largest positive value;
 *         -1 if all elements are negative.
 */
private static double positiveMax(double[][] m) {
  double max = m[0][0];
  for (double[] col : m) {
    for (double v : col) {
      if (v > max) {
        max = v;
      }
    }
  }
  return max;
}
```

26. Method that fills `board` with alternating black and white colors in a checkerboard pattern.

```
/**
 * Method that fills the given array of
 * <code>Color</code> objects with alternating
 * black and white colors in a checkerboard pattern.
 */
public void fillCheckerboard(Color[][] board) {
  for (int i = 0; i < board.length; i++) {
    for (int j = 0; j < board[i].length; j++) {
      if ((i - j) % 2 == 0) {
        board[i][j] = Color.WHITE;
      } else {
        board[i][j] = Color.BLACK;
      }
    }
  }
}
```

27. Method to test if one array "covers" another.

```
/**
 * Method to test if given array "covers" the other.
 * One array is considered to "cover" another if each
 * element is greater than the corresponding element
 * for at least half of all the elements in m1.
 * @exception IllegalArgumentException if given arrays
 *            are not of equal dimensions.
 * @return true if above conditions are met;
 *         false if above conditions are not met.
 */
private static boolean covers(double[][] m1, double[][] m2) {
  if (m1.length != m2.length) {
    throw new IllegalArgumentException("Given arrays are not of equal size.");
  }

  int count = 0;
  int elements = 0;
  for (int i = 0; i < m1.length; i++) {
    if (m1[i].length != m2[i].length) {
      throw new IllegalArgumentException("Given arrays are not of equal size.");
    }
    for (int j = 0; j < m1[i].length; j++) {
      if (m1[i][j] > m2[i][j]) {
        count++;
      }
      elements++;
    }
  }

  return (double) count / elements >= 0.5;
}
```

28. Method to produce Pascal's Triangle.

```
/**
 * Method to produce Pascal's Triangle with given
 * number of rows.
 * @return a two-dimensional array of integers
```

```java
 *         representing Pascal's Triangle with <code>n</code> rows.
 */
public int[][] pascalTriangle(int n) {
  int[][] triangle = new int[n][];
  for (int i = 0; i < triangle.length; i++) {
    triangle[i] = new int[i + 1];

    for (int j = 0; j < triangle[i].length; j++) {
      try {
        triangle[i][j] = triangle[i - 1][j - 1] + triangle[i - 1][j];
      } catch (ArrayIndexOutOfBoundsException e) {
        triangle[i][j] = 1;
      }
    }
  }
  return triangle;
}
```