Practice Problems

For each problem, produce a test suite using each strategy, separately:

a) specification-based testing (partitions and boundaries)

b) structure-based testing exclusively (MC/DC for the cases with multiple conditions, branch + loop coverage for the rest)

c) property based testing.

1. `sum`

```java
/**
 * This method takes two integers and returns their sum.
 * The numbers have to be between 1 and 99 (both inclusive).
 *
 * @param n and m integers
 * @return their sum
 */
public int sum(int n, int m) {
    if (n < 1 || n > 99 || m < 1 || m > 99) {
        throw new IllegalArgumentException();
    }
    return n + m;
}
```

2. `lastIndexOf`

```java
public static final int INDEX_NOT_FOUND = -1;
/**
 * Finds the last index of the given value in the array starting at the given
 * index.
 *
 * <p>
 * This method returns {@link #INDEX_NOT_FOUND} ({@code -1}) for a {@code null}
 * input array.
 *
 * <p>
 * A negative startIndex will return {@link #INDEX_NOT_FOUND} ({@code -1}). A
 * startIndex larger than the array length will search from the end of the
 * array.
 *
 * @param array
 *            the array to traverse for looking for the object, may be
 *            {@code null}
 * @param valueToFind
 *            the value to find
 * @param startIndex
 *            the start index to traverse backwards from
 * @return the last index of the value within the array,
 *         {@link #INDEX_NOT_FOUND} ({@code -1}) if not found or {@code null}
 *         array input
 */
public static int lastIndexOf(final int[] array, final int valueToFind,
                                                        int startIndex) {
    if (array == null) {
        return INDEX_NOT_FOUND;
    }
    if (startIndex < 0) {
        return INDEX_NOT_FOUND;
    } else if (startIndex >= array.length) {
        startIndex = array.length - 1;
    }
    for (int i = startIndex; i >= 0; i--) {
        if (valueToFind == array[i]) {
            return i;
        }
    }
    return INDEX_NOT_FOUND;
}
```

3. zigzag

```java
public static final int INDEX_NOT_FOUND = -1;
/**
 * This method receives a string s and a number of rows numRows,
 * and writes it down in a zigzag pattern. For example,
 * for s="PAYPALISHIRING", and numRows=4, the function returns
 * P  I  N
 * A LS IG
 * YA HR
 * P  I
 */
public String zigzag(String s, int numRows) {
    // some pre-condition check
    if(s.length() < 1 || s.length() > 1000)
        throw new IllegalArgumentException("1 <= s.length <= 1000");
    if(numRows < 1 || numRows > 1000)
        throw new IllegalArgumentException("1 <= numRows <= 1000");

    // early return: if the number of rows is 1, then, we return the same string
    if (numRows == 1) return s;

    // We create a list of strings, based on the number of rows we need
    List<StringBuilder> rows = new ArrayList<>();
    for (int i = 0; i < Math.min(numRows, s.length()); i++)
        rows.add(new StringBuilder());

    int curRow = 0;
    boolean goingDown = false;

    // We visit character by character, and we put it in the list of strings.
    // We change directions whenever we reach the top or the bottom of the list.
    for (char c : s.toCharArray()) {
        // add the letter
        rows.get(curRow).append(c);

        // are we at the top or the bottom of the list?
        boolean topOrBottom = curRow == 0 || curRow == numRows - 1;

        // add spaces if we are 'zagging'
        if(!goingDown && !topOrBottom) {
            for(int i = 0; i < rows.size(); i++) {
                if(i!=curRow)
                    rows.get(i).append(" ");
            }
        }

```

```
        // invert the direction in case we reached the top or the bottom
        if (topOrBottom) goingDown = !goingDown;

        // go to the next current row
        curRow += goingDown ? 1 : -1;
    }

    // we return the final string by simply combining all
    // the stringbuilders into a single string
    return rows
            .stream()
            .map(x->x.toString().trim())
            .collect(Collectors.joining("\n"))
            .trim();
}
```

4. `isItSummer`

```java
/**
 * This method predicts whether it is summer.
 * If at least 75% of the temperature values provided are 20 degrees
 * or above, it is summer. Otherwise, it is not summer.
 *
 * @param temperatures The list of temperature values
 * @return the probability of it being summer
 */
public static boolean isItSummer(List<Double> temperatures) {
    int count20OrAbove = 0;

    for (Double temp : temperatures) {
        if (temp >= 20) {
            count20OrAbove++;
        }
    }

    return count20OrAbove >= temperatures.size() * 0.75f;
}
```