PUZZLE (STOLEN FROM ATHINKINGAPE.COM)

Imagine you have a circle of people and you go around the circle removing every second person until one person is left. If you have 3 people in the circle, then the 3rd person will be the last one remaining. If you have 4 people then the 1st person will be the last one remaining. If you have 11 people then the 7th person will be the one remaining. If you have N people in the circle, who will be the last one remaining? Show off your work.

First, I created a quick Java algorithm to be able to see if I find some quick patterns:

```
public class PuzzleHelper {
	private final static int SIZES = 300; //Try group sizes from 1 to SIZES
	public static void main(String[] args) {
		for(int i = 1; i < SIZES; i++) {
			System.out.println("Group size: " + i + ". " +
					"Last standing: " + lastPerson(i));
	}

	/* Find the latest person left in the circle given a circle of size n: */
	public static int lastPerson(int n) {
		if (n <= 0) return 0; //Invalid input
		boolean[] circle = new boolean[n]; //true if removed
		int index = -1;
		while (n > 1) {
			index = nextAvailable(index, circle); //Jump one.
			index = nextAvailable(index, circle);
			circle[index] = true; //remove the person
			n--;
		}
		return lastStanding(circle);
	}

	/* Get the next person available starting from a given index: */
	private static int nextAvailable(int index, boolean[] circle) {
		do {
			index = next(index, circle.length); //index++ in the circle
		} while (circle[index]); //ignore those who were already removed
		return index;
	}


	/* Get the next index in a circular array: */
	private static int next(int index, int size) {
		return (index + 1) % size;
	}

	/* Return the first true boolean in the array: */
	private static int lastStanding(boolean[] circle) { //O(n)
		for (int i = 0; i < circle.length; i++)
			if (!circle[i]) return i + 1;
		return 0; // No one left :(
```

```
        }
}
```

And yes, there is a pattern. It's a sequence of even numbers that grows.

1 1 3 1 3 5 7 1 3 5 7 9 11 13 15 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 1 3 5 7 …

Let f(N) be a function that returns a list of the first N odd numbers. For example, f(3) returns 1, 3, 5. Thus, the result is the same as f(1) , f(2), f(4), f(8), f(16), f(32), f(64), … That means the size of the sequence is doubling!

Since it's doubling, we can take the lg(N). If lg(N) is a whole number, then the person to be eliminated is the first one. Otherwise, we can simply remove the biggest multiple of 2 from N and then get the Kth odd element where K is what was left.

Thus, one answer for the challenge is the *NthOdd*(n - *largestMultiple2*(n)).

Here's a Java implementation of the solution:

```
public static int lastPerson2(int n) {
        return getNthOdd(n - largestMultiple2(n));
}

private static int largestMultiple2(int n) {
        int largest = 1;
        do
                largest *= 2;
        while (largest <= n);
        return largest /= 2;
}

/* Get the Nth odd number of a sequence: */
private static int getNthOdd(int n) {
        return n * 2 + 1;
}
```

To make sure this formula gives the right solution, I've compared it with my previous algorithm for 3000 group sizes. They all match. Here's my test in Java:

```
private final static int SIZES = 3000; //Try group sizes from 1 to SIZES
public static void main(String[] args) {
        for(int i = 1; i < SIZES; i++) {
                int firstAlgOutput = lastPerson(i);
                int secondAlgOutput = lastPerson2(i);

                if (firstAlgOutput != secondAlgOutput) {
                        System.out.println("Houston, we have a problem.");
                        break;
                }
        }
}
```