题目描述：

给一个正整数列 nums，一个跳数 jump，及幸存数量 left。运算过程为：从索引为 0 的位置开始向后跳，中间跳过 J 个数字，命中索引为 J+1 的数字，该数被敲出，并从该点起跳，以此类推，直到幸存 left 个数为止。然后返回幸存数之和。

约束：

1）0 是第一个起跳点。

2）起跳点和命中点之间间隔 jump 个数字，已被敲出的数字不计入在内。

3）跳到末尾时无缝从头开始（循环查找），并可以多次循环。

4）若起始时 left>len(nums) 则无需跳数处理过程。

```
/**
* nums: 正整数数列，长度范围 [1,10000]
* jump: 跳数，范围 [1,10000]
* left: 幸存数量，范围 [0,10000]
* return: 幸存数之和
*/
int sumOfLeft(int[] nums,int jump,int left)
```

补充说明：

示例 1

输入：

```
[1,2,3,4,5,6,7,8,9],4,3
```

输出：

```
13
```

说明：

从 1（索引为 0）开始起跳,中间跳过 4 个数字,因此依次删除 6,2,8,5,4,7 。 剩余 1,3,9,返回和为 13

```
import java.util.*;


public class Solution {
    /**
     * 计算幸存数之和
     *
     * @param nums int 整型一维数组 正整数数列，长度范围 [1,10000]
     * @param jump int 整型 跳数，范围 [1,10000]
     * @param left int 整型 幸存数量，范围 [0,10000]
     * @return long 长整型
     */
    public long sumOfLeft(int[] nums, int jump, int left) {
        long sum = 0;
        for (int num : nums) {
            sum += num;
        }
        ArrayList<Integer> numbers = new ArrayList<>();
        for (int num : nums) {
            numbers.add(num);
        }
        for (int i = (jump + 1) % numbers.size(); numbers.size() > left; i = (i + jump) % numbers.size()) {
            sum -= numbers.get(i);
            System.out.println(numbers.get(i));
            numbers.remove(i);
        }
        return sum;
    }
}
```