1. public static int b(int n) { return b(n); }
   This call will keep returning itself, liso n over and over again until it crashes
2. Stack trace:maxArray
   int[] arr= {1,5,9,2};
   int max = *maxArray*(0, arr);

   Base case: when n==arr.length-1, meaning that the recursion has reached to the end of the array

   Else:

Stack	Value of arr[n]	Return (push)	Pop (bottom up) And int value returned
maxArray(1, arr)	1	Math.*max*(arr[0], *maxArray*(1, arr)) → Math.max(1, *maxArray*(1, arr))	Math.max(1, 9) → 9
maxArray(2, arr)	5	Math.*max*(arr[1], *maxArray*(2, arr)) → Math.max(5, *maxArray*(2, arr))	Math.max(5 ,9) → 9
maxArray(3, arr)	9	Math.*max*(arr[2], *maxArray*(3, arr)) → Math.max(9, *maxArray*(3, arr))	Math.max(9, 2) → 9
maxArray(3, arr)	2	Base case: *n=3=arr.length-1: *maxArray*(3, arr): Return arr[n] → return arr[3] → 2	*maxArray*(3, arr) Return arr[3] → 2

   Therefore, the method call returns 9

3. Stack trace:sumEvenNegative
   int[] arr= {2,3,6,-4};
   int a = *sumEvenNegative*(3, arr);
   N is initialized as size of arr-1

   Base case: when n==0: the recursion has reached the last element
   Test: if this last element satisfies the condition even or negative,
   then decide whether to add this last element to the sum
   Else:

Stack	Value of arr[n]	return(push)di	Pop(bottom up)
sumEvenNegative (2, arr)	-4	Passes if: arr[3]+*sumEvenNegative*(2, arr) → -4+*sumEvenNegative*(2, arr)	
sumEvenNegative (2, arr)	6	Passes if: arr[2]+*sumEvenNegative*(1, arr) → -4+6+*sumEvenNegative*(2, arr)	
sumEvenNegative (1, arr)	3	Goes to else: *sumEvenNegative*(0, arr) → -4+6+*sumEvenNegative*(1, arr)	-4+6+2=4
sumEvenNegative (0, arr)	2	Base case: n==0 Passes if in base case: %2==0 *sumEvenNegative*(0, arr) → Arr[n] → arr[0] → 2	Arr[n] → arr[0] → 2

Therefore, the method call returns 4

4.