Nice explanation. I have a much easier intuition which does not use 2 or 3.  
We could keep all the positive numbers 'balanced' to maintain the max product, which means the difference between each other should not be more than 1.  
Like 10 = 3 + 3 + 4, balanced; 10 = 3 + 2 + 5, unbalanced.  
Think about this case, x + y = n (x >= y and x - y <= 1) and the product is x \* y.  
If they are not 'balanced' which becomes (x + 1) + (y - 1) = n (note: (x - 1) + (y + 1) = n is still 'balanced') and the product is (x + 1) \* (y - 1) = x\*y - (x - y) - 1. Obviously, it's less than the one of 'balanced' case.  
So, suppose we could divide the number into k parts, n/2 >= k >= 2. For each k, num1 = n / k is the number for 'balanced', and for first n % k items, add 1 to it.  
Thus, we have (k - n % k) num1 and (n % k) (num1 + 1). Compare the product for each k to find the max.

public int integerBreak(int n) {

if (n == 2)

return 1;

if (n == 3)

return 2;

int max = 1;

for (int i = 2; i <= n / 2; i++) {

int num1 = n / i, num2 = n / i + 1;

int count1 = i - n % i, count2 = n % i;

int part1 = 1, part2 = 1;

while (count1 > 0) {

part1 \*= num1;

count1--;

}

while (count2 > 0) {

part2 \*= num2;

count2--;

}

max = Math.max(max, part1 \* part2);

}

return max;

}