Problem 4

1. I changed the constructor to set this.volume instead of volume and this.color instead of color because I think it would just be setting the parameter to itself otherwise due to local variables having priority over variables with a larger scope. Then, I had getVolume and getColor return volume and color instead of returning 0 and null.

2. I think keeping track of the volume and updating it whenever balls are added/removed is the better approach because it's more time-efficient. If the volume was to be calculated every time getVolume was called, that would be an $O(n)$ operation (n = number of balls). If the volume gets updated whenever balls are added/removed, then that's just an $O(1)$ operation each time (only adding/subtracting a number from another number). Then getVolume just returns a number which is another $O(1)$ operation. The only downside of storing the volume is that it uses slightly more space, but because it's just a number which shouldn't take up too much space, I believe it's worth the tradeoff.

3. 2 different ways of implementing getBallsFromSmallest:

    1. Make a copy of the underlying set of ballContainer (iterate through the set and add it to a list) and sort it by volume. Then return an iterator pointing to the beginning of the list as an unmodifiable list.

    2. Store a second set in Box that will keep track of all the balls sorted by volume. Every time a ball is added/removed, the ball should also be added/removed from this second set. Then, getBallsFromSmallest should just return an iterator pointing to the beginning of this set.

    I think the first option is better because while the runtime of getBallsFromSmallest will be worse ($O(nlogn)$ vs $O(1)$), the space being used will be better (no change vs doubling the number of balls being stored). I don't think the tradeoff in runtime of this one function is worth significantly increasing the amount of space being used by this data structure.