1. What is wrong with Ball.java?

    getColor and getVolume returned literals instead of the class's private color and volume variables.
    Solution: The return values for getColor and getVolume are changed to color and volume, respectively.

2. There are two obvious approaches to implementing getVolume():
    - Every time getVolume() is called, go through all the Balls in the Set and add up the volumes. **Hint:** one solution might be to use a for-each loop to extract Balls from the Set.
    - Keep track of the total volume of the Balls in BallContainer whenever Balls are added and removed. This eliminates the need to perform any computations when getVolume() is called.

    Which approach do you think is the better one? Why?

    The 1st approach creates O(n) operation for 1 function (getVolume).
    The 2nd approach creates O(1) operation for each of three functions (remove, add, and getVolume).
    Thus, the 2nd approach is better since it has better overall runtime.

3. There are many ways to implement getBallsFromSmallest(). Briefly describe at least two different ways. Your answers should differ in the implementation of Box, not in lower-level implementation (for example, using an insertion sort instead of a selection sort is a lower-level implementation because it does not affect how Box is implemented). Hint: think about different places in the Box class where you could add code to achieve the desired functionality.

   1. Making "contents" from BallContainer a sorted set
      - Specifically, a TreeSet
      - Comparator keeps TreeSet sorted by ascending volume
         i. Sorting is automatically applied when adding/removing elements
      - Duplicates automatically removed
      - getBallsFromSmallest() simply returns an iterator with iterator()
         i. TreeSet iterator iterates over the elements in ascending order

   2. Making "contents" from BallContainer a list
      - Specifically, a Vector for automatic resizing
      - Comparator keeps Vector sorted by ascending volume
         i. Add function first adds element, then calls Collections.sort on the Vector
         ii. Collections.sort uses the Comparator
      - Duplicates are checked with "contains" function before adding
      - getBallsFromSmallest() simply returns an iterator with iterator()
         i. iterates from start of list, which is already sorted in ascending order

Which of the above ways do you think is the best? Why?

Method 1 is better because it has better runtime. Inserting into a sorting tree (assuming use of proper search tree algorithms) has better runtime than appending into a list that is sorted after.