Joshua Collins

Principles of Software

Section 1

1/22/2024

Question 4: Getting a Real Taste of Java — Balls and Boxes

1. What is wrong with Ball.java?

   The values for the balls volume and color are not being saved in the object class. To fix this, I first changed the lines "volume = volume;" and "color = color;" to "this.volume = volume;" and "this.color = color;". Then, just to play safe, I changed the return statements on the getVolume and getColor methods from "return volume;" and "return color;" to "return this.volume;" and "return this.color;".

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?

   I think the approach of tracking the total volume via an integer in the public BallContainer class would be the best overall choice. by removing or adding a ball to the set named "Contents," for each change to the list, you only have to add or subtract the change in weight. This would be much faster than the for loop, since we don't add up all the ball volumes individually every time we add/remove a ball. This would save a lot of time if the set was large.

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.

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

   The first way to implement getBallsFromSmallest(), which is the way I did the method, was to use a list to store all of the balls in BallContainer. Once the list is complete, I sorted the list using a comparator which sorts via the volume of the balls. Once the list is sorted, you create an iterator to the list and that is what the method returns.

Another way to implement getBallsFromSmallest() is to use a tree hash set rather than the standard hash set that BallContainer uses. This would allow us to sort the set as we add or remove each ball to the box. This would automatically sort the set, so getBallsFromSmallest() would simply need to create and return an iterator for the set. The only downside is that each of these operations would be O(log n) compared the O(1) for hash sets.

To determine which method is better, we must determine how many times we call the method getBallsFromSmallest() as we add to the box. If we are only sorting once, the comparator method is faster since you only sort once and hash set operations are faster than tree hash set operations. Although, if we call getBallsFromSmallest() several times and change the box's contents between calls, sorting as we go would be faster. Thus, if we are changing the box's contents but want to sort it several times, the tree hash set would be more optimal. Overall, I would say the comparator method would be best for most situations since adding to the set is faster and the single sort would still be faster than sorting as you go via tree hash sets.