HW0 Problem 4: Getting a Real Taste of Java — Balls and Boxes

Jordan Ye

January 18, 2025

Problem 4.1: Creating a Ball

In the original `Ball.java` file, I identified two main issues:

- The constructors did not correctly assign the `volume` and `color` parameters to the instance variables. For example, writing

  ```
  volume = volume;
  color = color;
  ```

  simply reassigns the parameters to themselves rather than to the instance fields.

- The accessor methods `getVolume()` and `getColor()` returned fixed values (0 and `null`) instead of the actual stored values.

The fix was to use the `this` keyword so that the parameters are properly stored in the object. The corrected code for these parts is:

```
public Ball(double volume, Color color) {
    this.volume = volume;
    this.color = color;
}

public double getVolume() {
    return volume;
}

public Color getColor() {
    return color;
}
```

Additional Improvements: Error Handling in the Ball Constructor

The assignment included a test (`testCreateWithInvalidStringVolume()`) where an invalid string such as `"2o.4e2"` is passed into the constructor. Originally, the code was written to catch a `NumberFormatException` and throw an `IllegalArgumentException`. However, as pointed out in submitty discussion, throwing an exception causes client code (for example, on Submitty) to break—especially if the specification does not require the client to handle exceptions.

To fix this, I modified the constructor so that instead of throwing an exception, it silently sets the volume to a default value (0.0) when the string is invalid. This prevents the client from breaking, and the behavior in the case of an invalid string is acceptable per the assignment guidelines. The revised constructor is as follows:

```
public Ball(String volume, Color color) {
    try {
        this.volume = Double.parseDouble(volume);
    } catch (NumberFormatException e) {
        // Instead of throwing an exception, set volume to a default value.
        this.volume = 0.0;
    }
    this.color = color;
}
```

With this change, a call to `new Ball("2o.4e2", someColor)` will no longer throw an exception but will create a Ball with volume `0.0`. This behavior ensures that client code does not crash.

Problem 4.2: Using Pre-Defined Data Structures

For the `BallContainer` class, the task was to implement the following methods:

- `add(Ball)`
- `remove(Ball)`
- `getVolume()`
- `size()`
- `differentColors()`
- `areSameColor()`
- `clear()`
- `contains(Ball)`

Two approaches were considered for `getVolume()`:

- Recalculate the total volume by iterating over all balls every time the method is called.
- Maintain a running total that is updated whenever balls are added or removed.

I opted for the first approach—recalculating the total volume on each call—since the number of balls is relatively small, and this method avoids the complexity and potential bugs of keeping a running total synchronized with the container's state.

Problem 4.3: Implementing a Box

The `Box` class extends the functionality of a `BallContainer` by imposing a maximum volume limit. In addition to delegating standard methods to `BallContainer`, the `Box` class must implement:

- `add(Ball)` — which should only add a ball if the total volume does not exceed the maximum allowed.
- `getBallsFromSmallest()` — which returns an iterator over the balls sorted in ascending order by volume.

A straightforward implementation for `getBallsFromSmallest()` is to extract the balls into a list, sort it using `Collections.sort()` with a custom comparator comparing ball volumes, and then return an iterator over the sorted list. While an alternate approach would be to use a `TreeSet` to maintain sorted order automatically, this approach complicates handling duplicate entries and managing an additional data structure. Therefore, I opted for on-demand sorting, which leverages the Java Collections API without excessive complexity given the small dataset size.

The changes for the `Box` class are limited to enforcing a maximum volume during `add(Ball)` and providing a sorted iterator for `getBallsFromSmallest()`. The implementations of `Ball` and `BallContainer` remained unchanged.