6.092, Lab 1 Solutions

Note the variety of ways to implement BallContainer.add().

The term "control flow" refers to the execution's path through the program code. Suppose we have the following chunk of code:

```
if ( foo < 3 ) {
        doA();
} else {
        doB();
}
doC();
```

Suppose we run our program, and when we get to the above chunk of code foo is 2. In this case, we execute doA() and do not execute doB(). The if-statement creates a "fork in the road" of the program's control flow. The flow may come back together at doC(), since doC() will be executed regardless of the route taken through the if-statement.

A return statement immediately returns from the method--the compiler warns you against putting code after a return statement because that code will never get executed (so you probably made a mistake).

the add3() method is kind of slick, but it usually doesn't pay to be too slick. You might have to add a feature or fix a bug three months later and then have to spend energy refiguring out your code. Other people may have to read your code (a second pair of eyes helps tremendously with debugging), and they won't appreciate obsfucation (code that is hard for humans to read, though technically correct).

Next time you have to write a class that has a collection of objects that appear only once, consider using a Set! That way, you won't have to check whether contents already contain the ball or not. The less you have to remember to check the better--again, you might forget three months later when you're adding a feature, or someone else might be adding that feature and not realize your intentions. Comments are great, but no one can overlook things naturally working right.

Also, it is a good idea to use methods rather than directly referencing fields because the implementation of capacity may change later. Ultimately, we want to keep exactly one copy of any chunk of implementation, and have all places that use that notion reference that chunk. This is the same reason why we used UserInterface in lab2 instead of System.out.println(). We can easily implement a file i/o UserInterface and "plug" it into our MadLib code.

This is a non-crucial point in this context, but this is also why it might be preferable to return contents.size() instead of keeping track of the size everytime a Ball is added and removed. It's not just because the size() method is so handy (and cheap, computation-wise). Even without a handy List.size() method, computing something on the fly entirely in one place can be easier to debug and trust than code that depends on fields (which may be altered anywhere in the class file (or, if public, beyond)) being incremented and decremented right. There is something to be said for keeping the flow of information and dependencies compact. In fact, 6.170 is going to say a lot.

Observe how Box's add() overrides BallContainer's add(). In Box.add() we can reference getCapacity() without using the super keyword because Box does not override getCapacity(). However, we need super to reference the correct add--otherwise we'd get an infinite loop.

Finally, the Comparator in Box.getBallsFromSmallest is implemented as an anonymous inner class. It is anonymous because we did not instantiate a named reference to the Comparator, but instead instantiated and implemented it all in one go. Comparator is an interface with a single method, public int compare(Ball b1, Ball b2).

You can use an anonymous inner class anywhere you would normally provide a reference (variable name) to an object. Instead of giving the reference, write

```
new Foo() {
   ... implementation...
}
```

Notice the closing curly brace is inside the ');" that ends the call to the sort method.

Using classes in this fashion isn't crucial, but if you are only using the implementation once, like in this case, sometimes it's nice to have all the code in one place and not have to find the class file, BallComparator.java.

Finally, why can't the compare method simply return the difference between the capacities? Casting is part of the problem, but it is not the cause of the problem. What is the cause??