CS2230 Computer Science II: Data Structures
Homework 2
Objects in Java: the Construction site

Due February 5th, 2017, 11:59pm

Goals for this assignment

- Write a program that uses objects
- Finish an application consisting of more than one Java class
- Use arrays of objects
- Write an algorithm using an array of objects

Submission Checklist

Files

- ConstructionSite.java
- SandBox.java

Things to check for:

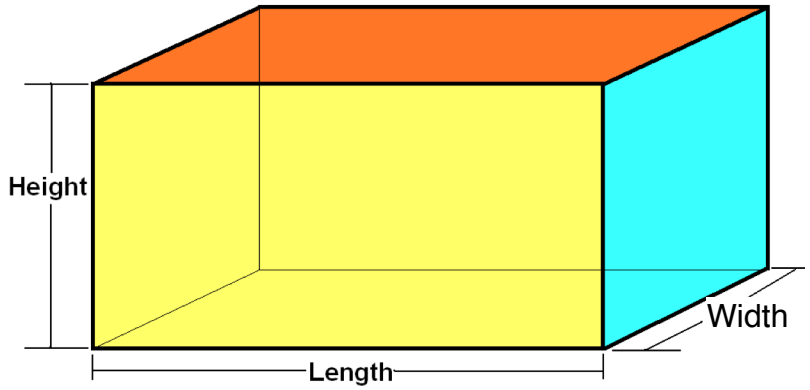
✓ Does ConstructionSite.java run properly?
✓ Did you double check your ICON submission to be sure you uploaded both of the Java files in the same submission?
✓ Did you re-download your submitted Java files to make sure they are the version you expected to upload?

Part 0: Setup the project in NetBeans

1. Unzip hw2.zip
2. Follow the same instructions as in Lab1 to create a new Java project from existing sources

Part 1: The SandBox class

We have provided starter code for a SandBox class in SandBox.java. A SandBox object represents a 3D SandBox shape, like a sand box (has an open top) with a length, width, and height.



A SandBox may be filled with sand up to its height. It could be empty, partially full, or completely full. As you'll see, we can scoop sand from one SandBox into another. After a SandBox is constructed, its dimensions never change again, but the height of the sand may be increased or decreased by scooping.

Your job is to provide the missing method bodies and methods. We suggest that you work in the following order:

1. Fill in the two constructors. One of them takes three arguments and the other takes zero arguments. You should follow the instructions provided in the comments for each one. In the comments, notice that SandBoxs must always start out empty (i.e., with no sand in them).

2. Write the methods `getLength`, `getWidth`, and `getHeight`. These methods do nothing more than return the value of the corresponding field.

3. Write the methods `getVolume` and `getSandVolume`. The methods return the volume of the SandBox and the volume of the sand in the SandBox, respectively.

4. We need a way to fill a SandBox if there is no other to pour from. Fill in the `fillToTop` method, which fills the SandBox completely, regardless of the original amount of sand in it.

5. We need a way to scoop sand from one SandBox into another. Fill in the `scoopSandFrom` method. This method takes another SandBox called `other` as an argument and moves sand ***from*** `other` ***into*** `this` SandBox.

Part 2: The ConstructionSite class

Now that we have working SandBoxs that can hold and scoop sand into, let's use several sand boxes to build a construction site. This construction site has a wily landscape architect. They are populating their site with lots of sand boxes of different sizes and shapes.

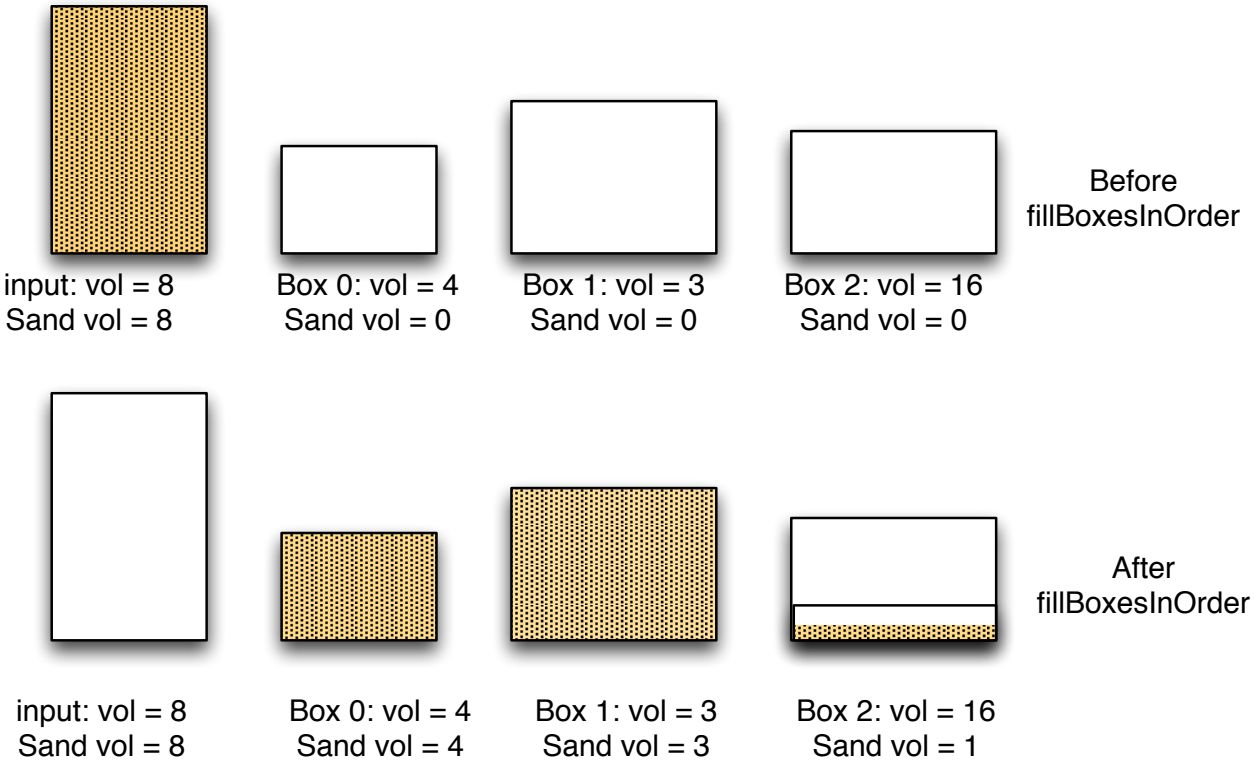
You will first add SandBoxs of different sizes to the ConstructionSite.

Second, we deliver sand to the site using a full "input" sand box. This "input" is a large SandBox containing all the sand in the simulation. We'll write a method for the situation where sand is scooped from "input" into the architect's SandBoxs in the order in which the SandBoxs were added to the ConstructionSite.

Notice that there can be many kinds of outcomes depending on the sizes of the input SandBox and the architect's SandBoxs.

Here is an example scenario. Brown is sand and white is emptiness. (we are not showing the 3rd dimension)

Example #1



To complete the `ConstructionSite` class, you should do the following.

6. We need a way to add SandBoxs to the site. Fill in the `addSandBox` method. Notice that we've already provided two fields for you to use to store the SandBoxs: a SandBox array called `SandBoxs` and an integer called `count`. If you need an example of how to add a new object to an array, see textbook chapter 3.1 or the lecture notes.

7. We need a way to delete SandBoxs from the simulation. Fill in the `deleteSandBox` method. It should delete the SandBox at the end of the `SandBoxs` array.

8. Next, you will fill in the `fillBoxesInOrder` method that scoops sand from the SandBox input to the architect's SandBoxs. Use the example above to make sure you understand the rules. The method must fill the SandBoxs in the order they were added to the site.

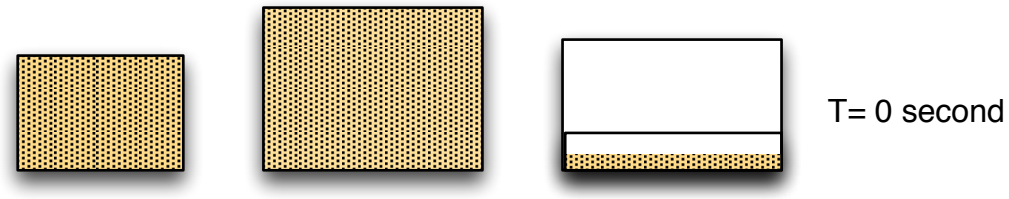
9. When you've finished the above parts, you should be able to run `ConstructionSite` to see if it works. There are three test cases provided (see the `main` method). Each test case gets a site, a sand input, and the *expected* volumes at the end of the game.

[See next page for the rest of the assignment]

Part 2b. Drain simulation

The fussy architect has decided that they want a different kind of sand. Since all the sandboxes are in place, they will drain the sandboxes by poking holes in them. Each hole will drain sand at a given volume per second. You'll simulate this situation in real time at a given *frame rate*. A frame rate of 2 would mean that every half second of real time you should calculate the new sand volumes of the sandboxes and print them to the screen. Here is an example of a simulation run at 1 frame per second.

Example #2

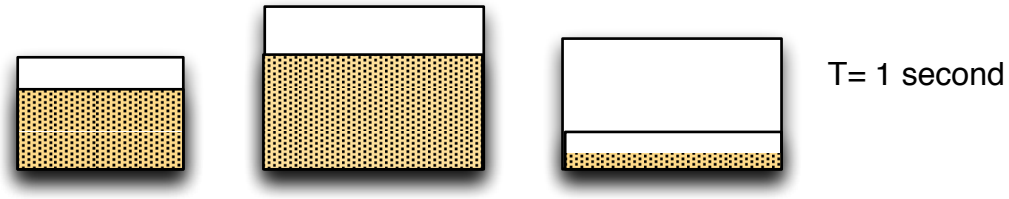


T= 0 second

Box 0: vol = 4
Sand vol = 4
hole = 1 unit/sec

Box 1: vol = 3
Sand vol = 3
hole=0.5 unit/sec

Box 2: vol = 16
Sand vol = 1
hole=0.25 unit/sec

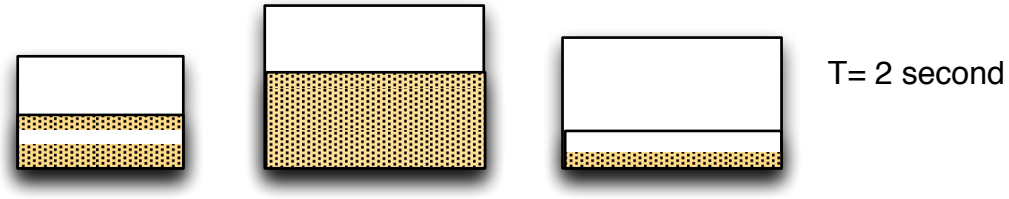


T= 1 second

Box 0: vol = 4
Sand vol = 3
hole = 1 unit/sec

Box 1: vol = 3
Sand vol = 2.5
hole=0.5 unit/sec

Box 2: vol = 16
Sand vol = 15.75
hole=0.25 unit/sec



T= 2 second

Box 0: vol = 4
Sand vol = 2
hole = 1 unit/sec

Box 1: vol = 3
Sand vol = 2
hole=0.5 unit/sec

Box 2: vol = 16
Sand vol = 15.5
hole=0.25 unit/sec

10. Complete the code in `runDrainSimulation`, such that it prints each box in `boxes` once per frame. Each frame should print **exactly one line**. See the end of

`expected.txt` for the required formatting. **You must do the draining by calling `scoopSandFrom`.**

Helpful Tips

- Does it mean my code works if I get the same output as what's listed in `expected.txt`? No, not necessarily! The tests we provided may not be sufficient evidence that your program is correct. We will test your submission on additional test cases. You can add more to the end of the main() method.
- What do I do if my program prints something like the following?

```
Exception in thread "main" java.lang.IllegalStateException: Sandbox is overflowed
        at hw2.SandBox.scoopSandFrom(SandBox.java:83)
        at hw2.ConstructionSite.fillBoxesInOrder(ConstructionSite.java:64)
        at hw2.ConstructionSite.testCase(ConstructionSite.java:94)
        at hw2.ConstructionSite.main(ConstructionSite.java:132)
```

This message means that you overflowed a SandBox, which should never happen. You need to debug your program. Try adding print statements to check whether the intermediate values of variables are what you expected. Use the line numbers (83, 64, ...) to tell you where in the program was when it stopped.

The above advice goes for any error message (NullPointerException, ArrayOutOfBoundsException, RuntimeException).

- Note that `scoopSandFrom` calls `checkState` to check physical properties of the SandBox. This method helps you find bugs more easily because it will often raise an error if you made a mistake filling or removing sand.