CS2020 Coding Test Mock Quiz

February 28, 2015

CODING QUIZ

Instructions

- Do not open the quiz until you are directed to do so.
- Read all the instructions first.
- The quiz contains 7 pages, including this one and the cover page.
- The quiz contains two multi-part problems. You have 120 minutes to earn 100 points.
- Please submit one .java file per problem. Please name your files as specified in the problem. Please compress your submission into a single .zip file containing your name.
- Each file should be submitted on IVLE before the end of the session.
- Document your code well. For each algorithm, explain your solution in comments. For each function/method, explain what it is supposed to do and how it does it. At least 10% of the points will be related to your documentation.
- Write your code clearly using good Java style. Points will be deducted for badly written code.
- Be sure to test your code for correctness. A correct, working solution will always get significant credit.
- All files are to be submitted within the **sg.edu.nus.cs2020** package.
- The quiz is closed book. You may bring one double-sided page of notes.
- You may also use any documentation available on the Oracle Java website, i.e. http://docs.oracle.com/javase/7/docs/api/. You may not use any other sites on the internet, your mobile phone, or any other electronic device. We may be monitoring the outgoing network connections to detect cheating.
- Read through all the problems before starting. Read the problems carefully.
- Do not spend too much time on any one problem.
- Good luck!

PROBLEM 1: THE COCONUT HUNTER (40 MARKS)

The durian king lives on an island where there are lots of coconuts. Interestingly, all coconuts in the island have the same weight. One day, the king was informed that there happens to be one coconut that weighs lighter than all other coconuts. The durian king has severe obsessive compulsive disorder. He cannot tolerate such imperfection. He wants you to remove the lighter coconut.

To aid you with your quest, the durian princess gave you a magical balance. You may only put equal number of coconuts on both sides of the balance, or else the balance will lose its magical power. The balance can accept any number of coconuts, and you can tell if one side is heavier than the other, or if both sides are of equal weight. However, the balance uses a lot of magical power, and you would like to minimise the number of times required to use the balance in the worst case.

For this task, you are required to create a new class called `CoconutHunter`. In this new class, implement the following method:

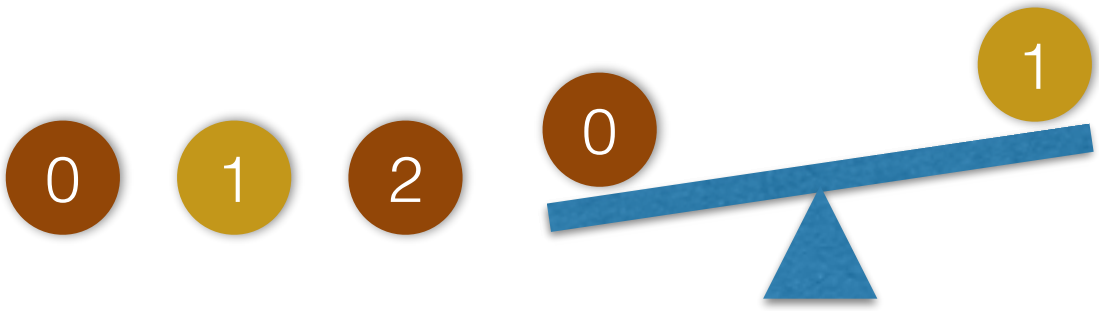
```
public static int findLightCoconut(BunchOfCoconuts coconuts)
```

which returns the index of the coconut which is lighter than the other coconuts in the `BunchOfCoconuts` class. The API for `BunchOfCoconuts` is provided as follows (you are not allowed to modify the contents of the class):

Constructor:

`BunchOfCoconuts(int n, int index):`

> Creates a collection of `n` coconuts in which the coconut located at `index` is lighter than other coconuts. Each coconut is labelled with a 0-based index.



Example 1: Coconut 1 is lighter than the other coconuts. Upon weighing coconut 0 and 1, we know that coconut 1 is the lightest coconut!

Methods:

Return Type	Method Description
`int`	`balance(List<Integer> left, List<Integer> right)` Takes in two lists that contain the indices of the coconuts placed on the left side and the right side of the balance respectively. Note that both lists need to be of the same size. Returns: -1 if the left side of the balance is lighter 1 if the right side of the balance is lighter 0 if both sides are of equal weight.
`int`	`getNumCoconuts()` Returns the number of coconuts in BunchOfCoconuts
`int`	`getNumBalance()` Returns the number of times balance is called.

Sample Execution

```
// Example 1 above
BunchOfCoconuts coconuts = new BunchOfCoconuts(3, 1);
ArrayList<Integer> list1 = new ArrayList<Integer>();
ArrayList<Integer> list2 = new ArrayList<Integer>();
list1.add(0);
list2.add(1);
coconuts.balance(list1, list2); // returns 1
System.out.println(findLightCoconut(coconuts));

BunchOfCoconuts coconuts = new BunchOfCoconuts(27, 12);
System.out.println(findLightCoconut(coconuts));
```

Sample Output

```
1
12
```

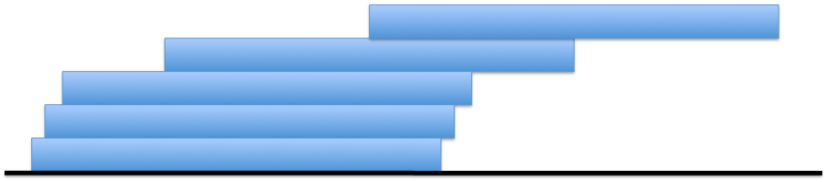
Remark

The number of coconuts ranges from 1 to 100,000. While your primary aim is to minimise the number of times the `balance` method is called, please make sure that the program terminates quickly enough in your implementation.

You may assume that all reasonable error checks has already been implemented in the `BunchOfCoconuts` class.

PROBLEM 2: BUILDING A TOWER (60 MARKS)

Imagine you have a very large supply of blocks, all of exactly the same size: 1 meter long and 1 kilogram in weight. (The depth and width do not matter for the purpose of this problem.) You want to use these blocks to build a tower, as in the following diagram:



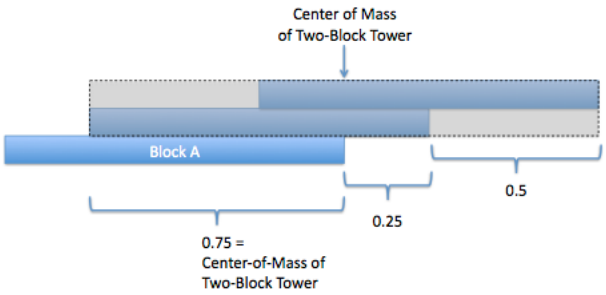
Your tower has exactly one block touching the ground, and extends as far as possible to the right. Your goal in this problem is to determine how far your tower extends if you have 1000 blocks.

We have provided you with a basic class `Block` which implements the `IBody` interface. A `Block` is a simple item defined by its weight, its length, and its centre of mass. The `IBody` interface supports some basic operations on physical bodies: it returns the centre of mass (as measured from the left endpoint of the body), the length of the body, and the mass of the body.

Part A (30 Marks)

Your first task is to complete the implementation of the `Tower` class which creates a tower from two existing `IBody` objects. The `Tower` class should implement the `IBody` interface. The constructor for the tower should take two `IBody` objects, along with a distance *d*. The second object is stacked on top of the first object, and the left edge of the second object is distance *d* from the left edge of the first object.

For example, consider the following stack of three blocks:



The centre of mass of the top block is distance 0.5 from its right endpoint, and hence it does not fall off. The centre of mass of the top two blocks is 0.75 from the right endpoint of the top block, and hence they do not fall of the bottom block. This stack can be constructed as follows:

```
Block a = new Block(1, 1);
Block b = new Block(1, 1);
Block c = new Block(1, 1);
Tower base = new Tower(a, b, 0.25);
Tower entire = new Tower(base, c, 0.75);
System.out.println(entire.getLength()); // Prints 1.75
System.out.println(entire.getCenterOfMass()); // Prints 0.8333333333333334
```

Hint: In order to calculate the centre-of-mass of two physical bodies, take the weighted average.

For example, if you are stacking object `o2` on top of object `o1`, and the left edge of `o2` is distance `d` from the left edge of `o1`, the the centre of mass of the combined `o1` and `o2` is found at:

```
((o1.centreOfMass * o1.mass) + ((d + o2.centreOfMass) * o2.mass)) / totalmass
```

where `totalmass = o1.mass + o2.mass`.

Notice that the `IBody` interface extends the `Iterable<Block>` interface. As such, the `Tower` class should implement `Iterable<Block>`. The iterator should return all the blocks that compose the tower from left to right, in order.

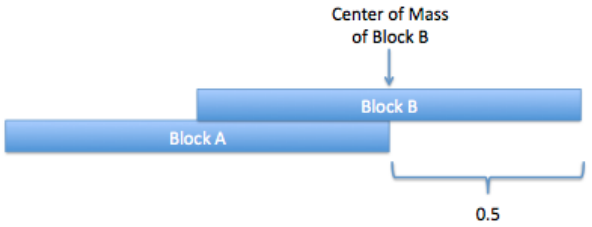
Part B (30 Marks)

Your second job is to implement a method static `buildTower(int numBlocks)` which returns a tower in the following manner:

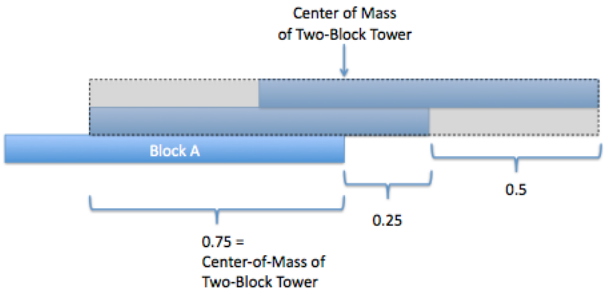
- A one-block-tower just consists of one block.
- A two-block-tower consists of a tower built from a single block with a one-block-tower on top.
- A three-block-tower consists of a tower built from a single block with a two-block tower on top.
- A n-block-tower consists of a tower build from a single block with a n–1-block tower on top.

Each time a new tower is built out of a block and an existing tower, the stack is arranged so that that the new tower is as far as possible to the right, without the tower toppling. That is, the centre-of-mass of the tower on top rests immediately over the right edge of the new block.

For example, the following is an example of a two-block tower:



The following is an example of a three-block tower:



If we execute `buildTower(1000)`, it should return a tower whose length is maximal (for towers stacked singly in this manner). What is this length?