CPSC 233 – Coding Challenge 2 – Practice 3

This is a practice coding challenge. You may ask other students and your TA any questions you wish. Note that you are expected to complete the actual coding challenge independently. Having a solution for this coding challenge will not help you come up with your own solution for the actual coding challenge. Only if you can solve this practice independently can you be confident that you can complete the actual coding challenge successfully. See previous coding challenges for rules for coding challenges and best practices for success.

Requirements

Remember to always create a 'skeleton' first and compile and test this using the provided JUnit test. For this coding challenge, you will be asked to create two classes. Make sure you create a skeleton of both classes first. WebCAT needs a version of both classes that compile with the tests placed in a zip file. If one of the classes is missing or does not compile, none of the tests will be run and all tests will be marked as failed.

Requirements for Skeleton of Ball and BallPit classes

	Ball	**BallPit**
Instance Variables	bounciness: double height: int	name: String ballList: ArrayList<Ball>
Constructors	1: arguments: bounciness, height 2: copy constructor	1. arguments: name
Getters	bounciness and height	name and ballList
Setters	bounciness and height	
Methods	numberOfBounces():int bounce():void equals(Ball):boolean	add(Ball): void getBallWithMostBounces():Ball

Additional Requirements for Ball Class

- The *height* of the ball should be greater than 0. If 0 is less is provided, set it to a default of 1.
- The *bounciness* is a percentage value (between 0 and 1) and should not be 0% (meaning the ball will not bounce) nor 100% meaning the ball always bounces back to the same height. If an invalid percentage is provided, set the bounciness to a default of 50%.
- The *bounce()* method should update the height to reflect a single bounce. The ball will be bounciness% of it's original height after a single bounce. (eg, if the height is 100 and bounciness is .9, then height of the ball is 90 after a single bounce.) The height should always be rounded down to the nearest integer.
- The *numberOfBounces()* method returns the number of bounces the ball will take before remaining still on the ground (height is zero). The height of the ball should not change as a result of this method.
- The *equals()* method takes a Ball as an argument and returns true if the argument ball has the same height and bounciness. Note that the operator == will return true if two objects refer to the same object.

Additional Requirements for BallPit Class

- All instance variables must be completely encapsulated. There should be no privacy leaks.
- *getBallWithMostBounces* returns the ball in ballList that will do the most bounces before being still on the ground. If the list is empty, this method will return null.
- Don't forget to import java.util.ArrayList

Notes

You'll likely create (while) loops to complete this exercise. If your code contains an infinite loop it will look like the test is doing nothing. To verify this is the case, comment out loops and run the tests again.