Question 1. (15 points) Recall that the decorator pattern allows you to attach additional responsibilities to an object dynamically. We discussed the DeceleratingBall example in lecture.

```java
public class DeceleratingBall extends MovableBall {
    private MovableBall workerBall;

    public DeceleratingBall( MovableBall aBall ) {
        super();
        workerBall = aBall;
    }

    public void move() {
        workerBall.move();
        workerBall.setMotion( workerBall.xMotion()*0.95, workerBall.yMotion()*0.95 );
    }

    public void paint( Graphics g ) {
        workerBall.paint( g );
    }

    // remaining methods just delegate to the workerBall like "paint"
}
```

a) Circle the declarations that are allowed in Java:
   i.  MovableBall ball = new DeceleratingBall( new BoundedBall(10, 15, 5, 2.0, 5.0, this) );
   ii. DeceleratingBall ball = new DeceleratingBall( new BoundedBall(10, 15, 5, 2.0, 5.0, this) );
   iii. DeceleratingBall ball = new DeceleratingBall( new CannonBall(10, 15, 5, 2.0, 5.0 ) );
   iv.  DeceleratingBall ball = new DeceleratingBall ( new PinBall (10, 15) );

b) What are some advantages of using the decorator pattern to implement a decelerating ball instead of writing subclasses DeceleratingBoundedBall, DeceleratingCannonBall, etc.?

c) What is the main disadvantage of using the decorator pattern?
d) Modify the DeceleratingBall code to include another constructor that is also passed an argument of type double that allows you to vary the deceleration factor to something other than 0.95. Write the new constructor below and modify the code on the previous page as necessary.

Question 2. (15 points) Add an ExpandingBall class to the MovableBall hierarchy. An ExpandingBall becomes a little bit larger every time it moves. Use the decorator design pattern technique like we used for the DeceleratingBall class. You do not need to write all the delegated methods.