NSFAQ :: PONG

This is a list of not so frequently asked questions. If you send me a question by Canvas message, I may post it here. I will always anonymize the questions as I don't want you to feel self conscious about asking me questions. However, many students have the same or similar questions.

First though, let me be clear, this is not intended to be a brain buster on first quarter physics. If you have questions about the basic physics, ask, I'll set you in the right direction. That's why I gave you the code-along video. This is a software engineering course, not a simulation coding course.

Q: My ball is going crazy sometimes when it intersects the bat, what's going on?

A: Almost certainly what's happening is that you are detecting a collision while there is an existing collision. If you set a Boolean value for relationship between the ball and that object and only process the collision when you aren't already colliding, then this should go away. Pseudocode follows:

```
if( ball is colliding with bat){
  if(!inCollision ){
     Do the things that I need to do when the ball collides with the bat
     inCollision = true;
  }
}
else if(ball is colliding with the wall){
  if(!inCollision){
    Do the things that I need to do when the ball collides with the wall
    inCollision = true;
  }
else if ...
else
   inCollision = false;
```

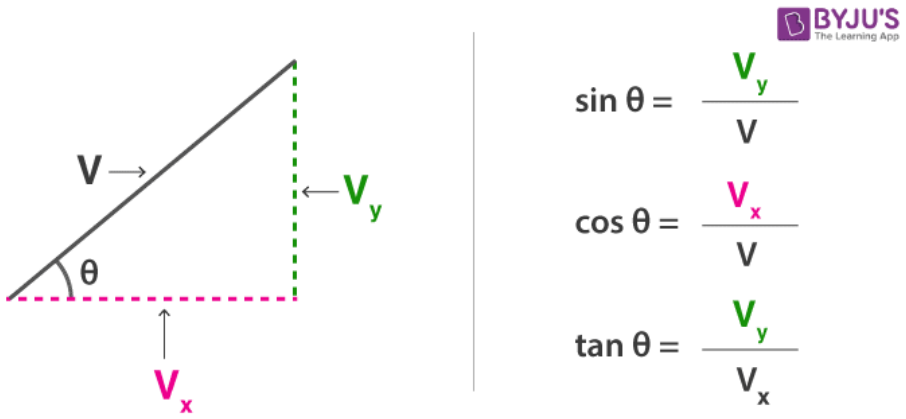
Notice how this sets the flag when you collide, then checks for collision, and sets the state so that the game knows that the ball is currently "in some collision" Now, when the ball no longer collides with the bat then the collision check returns false and the `inCollision` flag is cleared allowing the next collision to take place. Now, this is overly simplified and really only works because it's unlikely that the ball will collide with two objects at once. You can add more Boolean flags that are specific to different directions, or even different objects, so that it will still check other objects while "in collision." For example, you could replace `inCollision` with `inBat`, etc. In this case, you will just give each collision its own else clause and Boolean.

```
if( ball is colliding with bat){
  if(!inBat ){
    Do the things that I need to do when the ball collides with the bat
    inBat = true;
  }
}
else
  inBat = false;
```

Q: I'm having difficulty computing the various angles required for the ball direction, how do I do that?

A: You are overthinking this. A Ball has a horizontal and a vertical position and velocity. That is, the velocity has two components. As the ball is falling down the screen its velocity is positive as the origin of the screen is in the upper left hand corner. When the ball strikes the bat, the vertical velocity is in the opposite direction. In other words, it is negative. Similarly, when the ball hits the right side of the screen, the horizontal velocity is positive, when it goes back towards the left, its horizontal velocity is negative. In other words, striking a border changes the sign of either the horizontal or vertical velocity.

Ok, let's visualize this, here's an image from the web (https://byjus.com/physics/velocity-vectors/) :



Your velocity vector should have two components, and X component, and a Y component. The author of the code-along uses a Point2D for this. Well, it's not really a point, but whatever, that's fine. When the bat hits the ball, the Y component should get larger by some fixed amount that has nothing to do with the speed of the bat. Since the ball is moving downward at the time of the strike, its velocity should be positive, so you can just add a positive constant. Note that this is what we call *temporal coupling*. That is, your lines of code are not independent in time because you need to add the velocity value before you change the direction of the ball by changing the sign of the Y component. Obviously sequential code often has this property, what makes this temporal coupling is that the reasons are hidden in the data. This is the kind of thing that should be lifted to a method so that the method name can communicate to others what's going on:, e.g. `accelerateFallingBall(double amount)`. Think about how you could code the offset so as to eliminate the temporal coupling?

Additionally, the bat is also moving at some speed, you take some of that value and add it to your X component. If the bat is moving left to right, its speed will be positive, if the bat is moving right to left, its speed will be negative. Consequently, you can just add this signed value to the signed value of the ball's X velocity value and it will be self correcting. If the ball has positive horizontal velocity, smacking it the other direction will slow down its horizontal velocity.

Finally, you also need to rotate the ball, but the same thing holds here. You can just use some scaled value of the bat velocity and add it too the current rotation value. If the ball is rotating in a clockwise direction, a negative number will cause it to rotate more counterclockwise. You only need to call `setRotate()` and `getRotate()` on the ball, e.g., `ball.setRotate(ball.getRotate() + offset)`; where offset is the amount you want the rotation to change.

Notice that there are no angles, anywhere. Yet we have completely described the physics of the pong ball. If you're struggling with this then you are using too much coding from example and not enough coding from discovery. The physics are intentionally simple and all that you need to know is that velocity is a vector.

Q: I don't know what to say in my video, or how it's different from the writeup, can you give me more guidance?

A: They are not the same thing. You are creating a simple playable game within the constraints of a short assignment within a quarter. The writeup should be focused on why you made the big picture design choices. Why did you choose the object hierarchy? How did it facilitate *separation of concerns*, to what extent do you believe that it yielded *greater cohesion* or *looser coupling*? How do you think that it's more understandable and will make sense to other developers? On the other hand, the video is for you to tell me about your game and how you chose to make things work. How did you implement various features? Tell me about anything special in your game. Did you find that you had to limit some aspect to make it more playable, how did you do that? For example, "I found that I had to limit horizontal velocity as well as vertical velocity because otherwise the ball would go insane from right to left due to my high frame rate." Or, "I chose to scale velocity with the frame time so that my game gives roughly the same experience on any computer, fast or slow. " You should be excited to tell me about your game and give me small details on how you solved problems *while you are playing the game*.

Q: I'm really concerned that I didn't make the right decision and will lose points because my OOP isn't correct. Can you look at my code and see if it's right?

A: Well, no. But, let's pull back from focusing on the trees and try to see the forest. There's not a correct answer. Well, some answers are better than other answers but what we're looking for here is a good justification and discussion of the approach that you took. Use the language of OOP insofar as we have discussed it to date. Communicate to me in your own words and in technical language what your thought process was, how did you get to your final solution? The programming is not the entire picture here. Your understanding of OOP and design is what we want to see. Yes, your code will be graded and we're looking for sincere effort to adopt best practices. However, your design and development process, as discussed in your writeup, is what we're going to evaluate to see if you are getting the ideas. Don't dismiss one of the other, but don't spend too much time analyzing this to death, that is the anthesis of agile. Refactoring your design is an important part of the design process.

Q: For Pong-1 do II have to put everything in one method?

A: Gah! No! Use as many methods in the PongApp class as you like. In fact, you should be using this version to do a procedural decomposition to help you name the actions within the code. It does not all have to be in one method. Of course the sound class is a separate class. The fundamental idea here is that you don't want to create new classes for Bat, Ball, Score, etc., but there is no reason that you can't use factory methods to create them.

```
class PongApp extends Application{
  Rectangle batFactory(){
    Rectangle r = new Rectangle();
    // setup the r so it's a bat with whatever it needs.
    return r;
  }

  ...somewhere...in some method
     Rectangle bat = batFactory();
     Rectangle ball = ballFactory();
  ...do these really need to be different methods?
```

The key idea here is that you are solving all of the little problems in methods and procedural composition. Your main app can still largely be in one method, but, you can call other methods within the class. You could even use some small classes, but, Bat and Ball and other basic game objects should not be classes here so you can use your linear code to help guide you to what is the right object decomposition. What behaviors do you have that can be grouped together into a class? Is a batFactory necessary, or will a constructor do the job?

Q : How do I center the text in the score?

A: You could pre-render the text in the desired font within a different Text object and the get the width of that text, as shown here:

```
        Label someScoreLabel ...
```

```
        Text theText = new Text(<your string goes here>);
        theText.setFont(someScoreLabel.getFont());
        double width = theText.getBoundsInLocal().getWidth();
        someScoreLabel.setTranslateX(x-width/2);
```

Alternatively, depending on how you've coded the game, you can manipulate the layout properties of a text or label object directly. That is almost certainly cleaner and more elegant, but I don't think that it's easier.