Functionality: We don't get any compile time errors. We still do get the concurrent modification exception every once in a while which might cause our program to freeze. Even though we implemented the try catch statement.
Design: Our design is interesting, especially in terms of how the health system works with player and spheres. Firstly, we display the health of everything as integers. Then, these integers are displayed on the body of the spheres or player. The sphere's health depletes normally, but player's health depletes like a timer almost. Instead of being a set amount of times player can be hit like in standard games, Player can be in contact with enemies for only so long, which encourages movement. Also, the unpredictable bouncing element of the spheres will keep players on their toes.
Creativity: This game is not super creative. It takes elements from the game Ball Blast on the App Store and Bubble Trouble from Cool Math Games. It takes the displayed health element from Ball Blast and the moveable player from Bubble Trouble. However, the balls in those games bounced while ours traveled in straight lines.
Sophistication: Our program is relatively sophisticated. We raised the FPS to 120 to increase how smooth the game runs. We also used Timers and Thread.sleep in order to affect the game as the player plays. For example, once the player finishes the level, we use Thread.sleep to quickly pause the game to give a quick break to all the action. Also, we used file loading to save images to use as backgrounds for the game. Finally, we used some heavy math elements in our game, like the utilization of the distance formula to detect collisions between spheres colliding with themselves and spheres colliding with player.
Broadness: We ticked boxes 1, 2, 3, 5, 6, and 7. Here's where we used each and how it was justified:
1. We used ImageIO and Images in order to implement our backgrounds. We've never done something like this in lab, so this is a library from outside of class.
2. Bullets extend Spheres. Both classes have similar variables, but act differently. Spheres, for example, bounce of the walls in the Main class, but bullets do not. Bullets travel only vertically, but spheres have a vertical and horizontal component.
3. We have the interface Interactable, which basically states that anything that's interactable can be drawn and updated. This is justified because both Player, Spheres, and Bullets need these methods and it served as a good reminder to have to override these methods from Interactable.
5. We used the built in data structure Vector to contain our Spheres and Bullets. This is justified because we needed a infinitely growing list that could contain all of our spheres and bullets.
6. Our photos are inputted in the Main class.
7. We used randomization in many instances across the program. We randomized the spheres' sizes, speeds, and health values. This is justified because this level of randomness makes every playthrough of Burst Your Bubble different.
Code Quality: We commented all our code and followed correct bracket and conventional order of writing. Although we both coded and our style of coding is a bit

different we tried our best to keep the code consistent across the files. The product does look semi-professional with our use of backgrounds and title screen.