Starting off the project I began looking at any possible ways inheritance could be implemented into the code. I noticed right away that the Ball and Bat classes could become child classes to a GameObject class that would control all of the objects in the game. I began by importing all of the necessary variables into the GameObject class and defined most of them as protected floats. This allowed me to keep the variable public within the class and subclasses, but still have some data protection. I also made the GameObject class abstract in order to only allow inheritance to the child classes. Within the Bat and Ball classes I noticed that they had very similar movement update classes, and relied on similar variables. I decided to make an update class in the GameObject parent class that was primarily based on updating the balls movement. I then override this update class in the bat class in order to get the correct updates on the bat's movement. I also included many of the Ball's classes into the GameObject class since they could be used for all of the child classes of the GameObject class. I could have kept many classes in the Bat class, but ultimately decided to put them into the GameObject class in case other Objects needed the functionality in the future for any reason. This allowed me to use the reset class on the Bat as well and definitely cleaned up how the game restarted. With this increased functionality, I can also speed up or slow down whether or not the bat also speeds up as the game goes on. With this large change that I implemented, I was able to clean up the bat and Ball classes substantially, which made them much easier to read and understand. The GameObject class is still a little messy, but it definitely makes the code much more legible to read and much more efficient to understand for the Bat and Ball classes.

Next I went to the Pong Game class to try and decipher some of the ways I can clean the code up. My first instinct was to separate the sound loading and deployment from out of the main class into a separate class. This allowed me to create the SoundManager class which allows me to add all of the sound into the game in a clean and efficient way. In this class I decided to clump the try catch clause into the SoundBuilder method so that I only have to call one method in PongGame. Alternatively I could make the try catch in a separate method to simplify my code even further, but it would add some more clutter to the PongGame class. As a result of this code inheritance, I was able to reduce the clutter in PongGame and only require a new method creation and descriptor line of code to be added into SoundManager to add a new sound to the game. This greatly reduced the time it took to add sound into the game and greatly increased efficiency in understanding and displaying my code.

Lastly I continued to look through the PongGame class for any other ways to make the code easier to read. I noticed that there were a lot of methods that either ran the game, paused the game, started a new game, etc. So I decided to make a GameLogic class that would handle all of the changes to the game's state. I began by implementing the startNewGame method into the class and successfully moved it by copying the old code over and creating a new instantiation of mScreenX, mScreenY, Ball, and Bat. This helped improve the visibility somewhat, but I was not able to fully achieve my vision with this class. Some of the other methods such as pause and run would certainly fit into this class easily and clear up the PongGame class substantially, however I was not able to figure out how to properly move them into GameLogic.

Additionally I would've created a TextManager class that would host the draw method and Debugging method. This would have greatly reduced the amount of clutter in PongGame and made the readability of that class much better. Unfortunately I was not able to develop this class, but I like to pre plan as much as possible to improve the visibility of my code, and efficiency when I code.

Overall, I think my solutions to the code provided were extremely helpful and provided some clarity to the code. I would definitely begin earlier next time in order to fully understand and plan out how to simplify the code further. This was a great review of basic OOP.