1. Make an initial assessment of the project so far: does it compile and execute without error, what does it do, does it appear to function correctly based on the description above, is it well-commented, does the Javadoc give sufficient details, does the code look efficient, has it been tested fully?

The project compiles and executes without error. The project includes multiple objects that can be clicked on, and when they have been clicked the object is removed. The click position is a bit off, so that when the object is clicked it does not always get removed. This program does not appear to function correctly based on the description above, as there should be some fish behind these objects when they are clicked, however there are none. The commenting included in this project is fine but could be improved on by adding more descriptive comments, as some of them are a bit bare bone. However, the format for the comments are meeting the standard of the Javadoc, so that the method descriptors and class descriptors display in the Javadoc when it is generated. The use of keywords for commenting, for example ‘DECLARE’ and ‘INSTANTIATE’ are consistent across each class, making the code more readable to other programmers. The Javadoc has not been generated when the project was first opened. After regeneration of the Javadoc, the details are sufficient enough for the reader to understand the classes. There is no unit testing included with this project, so as far as I can see, very little testing has been established with this project.

1. Does the Game.FishSweeper class exhibit strong cohesion? Explain your answer.

In OOP, Cohesion refers to how a single class is designed and that it is design with a single well-focused purpose(Cohesion in Java - GeeksforGeeks, n.d.). Having a class with strong cohesion, allows that class to be maintained much more easily and changes to that class are required less frequently(Cohesion in Java - GeeksforGeeks, n.d.). Low cohesion is when a class does many different jobs that don’t have much in common(Cohesion in Java - GeeksforGeeks, n.d.).

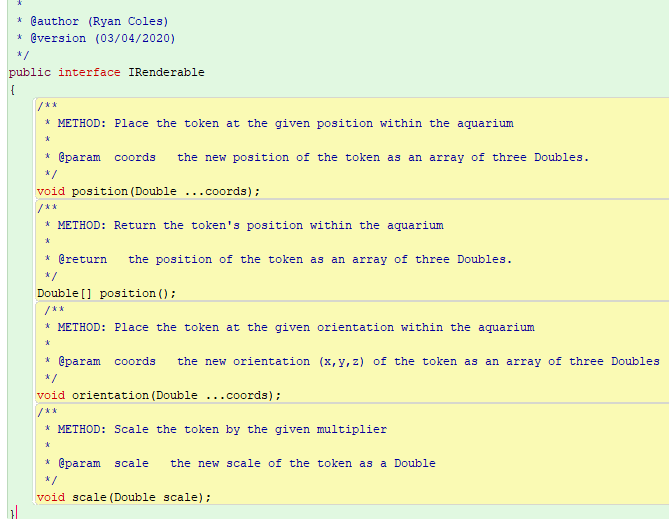
The FishSweeper class exhibits weak cohesion. I believe this is true, because the class has the job of populating the world with renderable objects, listening for mouse input, creating the world, destroying the game world and updating the game world. All these responsibilities are not very similar and could be spilt off into different classes to make the FishSweeper class more cohesive.

1. The relationship between Game.FishSweeper and Framework.Renderable violates the Dependency Inversion principle. Implement a modification to this relationship that resolves this issue.

The idea of the dependency inversion principle is to make sure that high-level modules don’t depend on low-level modules, instead these should both depend on abstractions(Janssen, 2018.). Following this principle will lead to your code being reusable and unaffected by any alterations made in low-level modules. The Abstraction that is required of this principle will decouple the high-level and low-level modules from each other because they are not depending directly to each other but instead depend on the abstraction of the modules(Janssen, 2018.). Due to the abstractions that dependency inversion creates and the different implementations of these abstract dependencies, natural extension points are put in place, which allow for the modification of behaviours without altering the class itself (Janssen, 2018.).



The FishSweeper class and Renderable class violate this principle because the FishSweeper class is depending on the Rederable class itself and not the abstraction of it. So a solution this problem would be have Renderable implement an interface which would be an abstraction of itself, then FishSweeper will only know about this abstraction of Renderable and will depend on the Renderable interface instead of the Renderable class.





After implementation if this new interface the project complies and executes as expected and the FishSweeper class is no longer dependent on the Renderable class.

1. A line of code towards the end of the ‘run’ method of the Game.FishSweeper class is commented-out, because it generates an exception at run-time. The line of code removes a Renderable instance from an ArrayList when the instance is no longer needed (ie the target was clicked on). Implement a change to the code that achieves this without generating a run-time exception.

References

GeeksforGeeks. n.d. *Cohesion In Java - Geeksforgeeks*. [online] Available at: <https://www.geeksforgeeks.org/cohesion-in-java/> [Accessed 3 April 2020].

Janssen, T., 2018. *SOLID Design Principles Explained: Dependency Inversion Principle With Code Examples*. [online] Stackify. Available at: <https://stackify.com/dependency-inversion-principle/> [Accessed 16 March 2020].