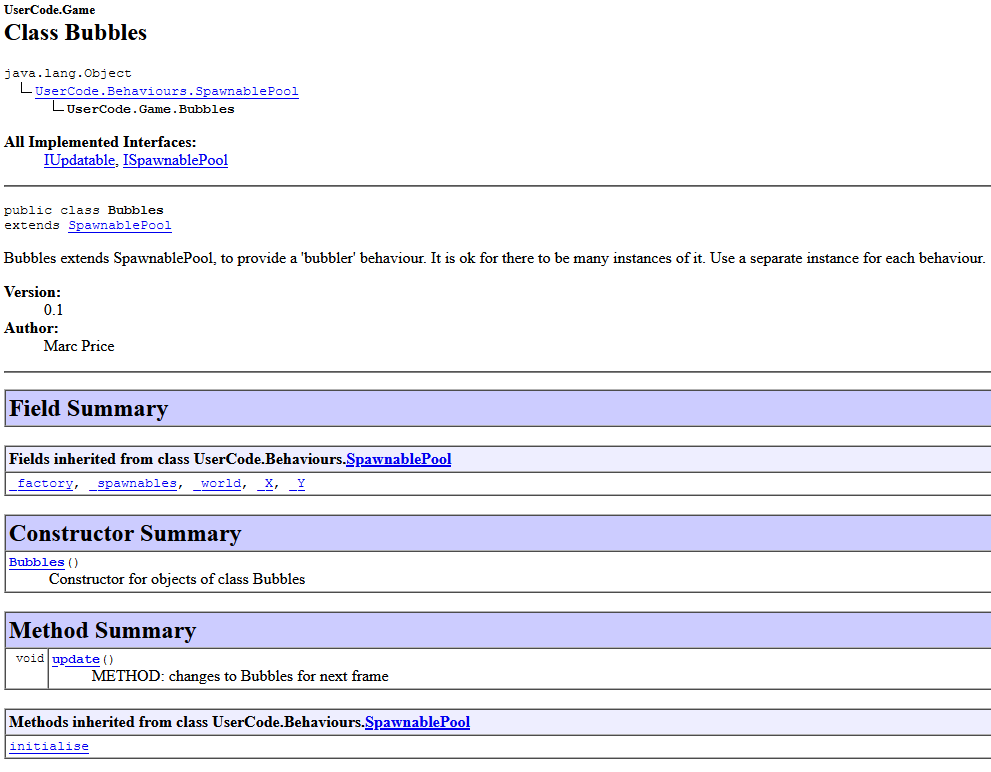
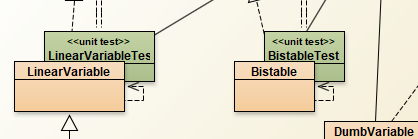
1. Make an initial assessment of the project so far: does it compile and execute without error, what does it do, does it appear to function correctly based on the description above, is it well-commented, does the Javadoc give sufficient details, does the code look efficient, has it been tested fully?

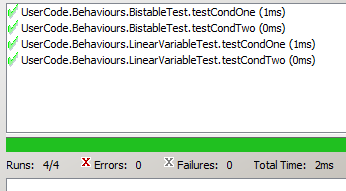
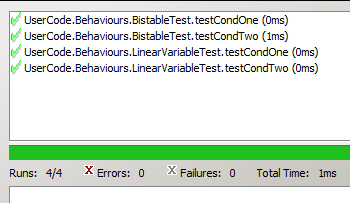
On first glance of this project, I can see that the program compiles successfully without error and mostly functions correctly based on the description above. The code is well commented, showing use of method descriptors, class descriptors.



Each element of the Javadoc is sufficiently detailed and very easy to follow.



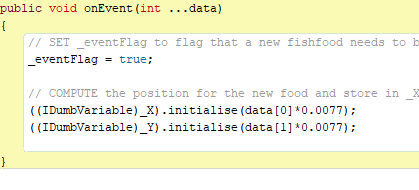
The project includes two unit tests in the behaviours package. Both tests are sufficiently commented and tested and include 4 tests each, which pass without error.

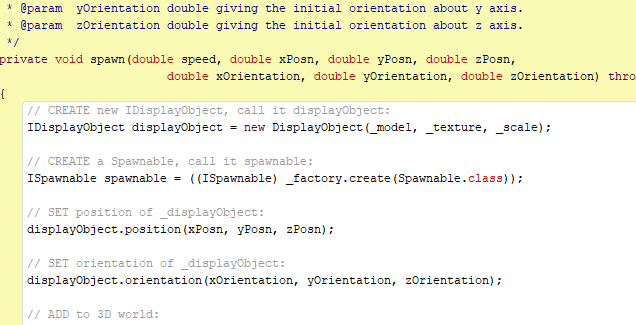


2.Does the Feeder class in the Game package exhibit strong cohesion? Explain your answer.

In Object Oriented Programming, Cohesion is the principle that refers to how a single class is designed. This class must be designed with a single, well focused purpose(Cohesion in Java - GeeksforGeeks, n.d.). The more focused a class is, the more cohesive it is(strong Cohesion) (Cohesion in Java - GeeksforGeeks, n.d.). The main advantages of a cohesive class are that it is easier to maintain and that is more reusable than classes with a less focused role (weak Cohesion) (Cohesion in Java - GeeksforGeeks, n.d.).

The feeder class does not exhibit strong cohesion. I believe this is true because, the feeder class is managing multiple roles. These roles are spawning the fish, listening for an event, and updating the fish. Not having a more focused role, for example just spawning the fish, makes it a weak cohesive class.



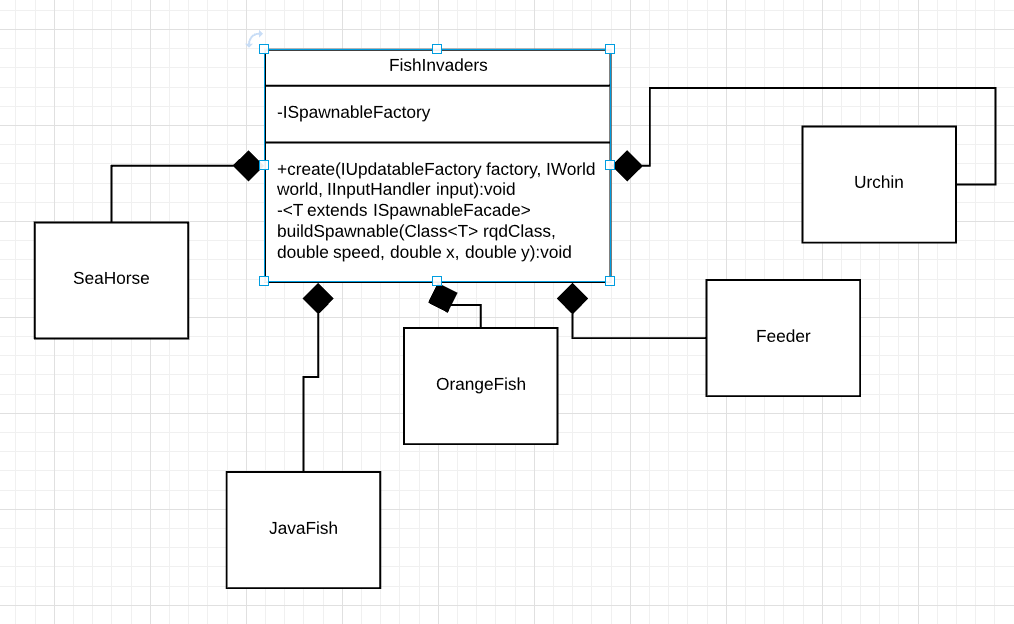


3.The relationship between Simulation and Game.FishInvaders violates the Dependency Inversion Principle. Implement a modification to the code that resolves this issue, and explain the changes you have made.

Dependency Inversion’s general idea is that

* *high-level modules shouldn’t depend on low-level modules. Both Should depend on abstractions(* Janssen, 2018.).*.*
* *Abstractions should not depend on details. Details should depend on abstractions* (Janssen, 2018.).

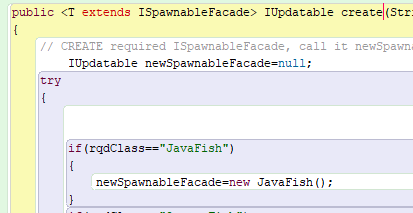
Following this principle will lead to your code being reusable and unaffected by any alterations made in low-level modules. The Abstraction that is required of this principle will decouple the high-level and low-level modules from each other because they are not depending directly to each other but instead depend on the abstraction of the modules(Janssen, 2018.). One key part to note about this principle is that it helps us respect all the other SOLID principles but primarily the Liskov substitution principle and the Open/Close principle(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.). This is a requirement of the Open/Closed principle and in turn almost enforces this principle naturally. Dependency inversion also makes you use subtyping in the correct manor where you are programming to the high-level module rather than the class itself, which follows the Liskov Substitution principle so that you can replace the implementations with others of the same interface without the code breaking(Janssen, 2018.).



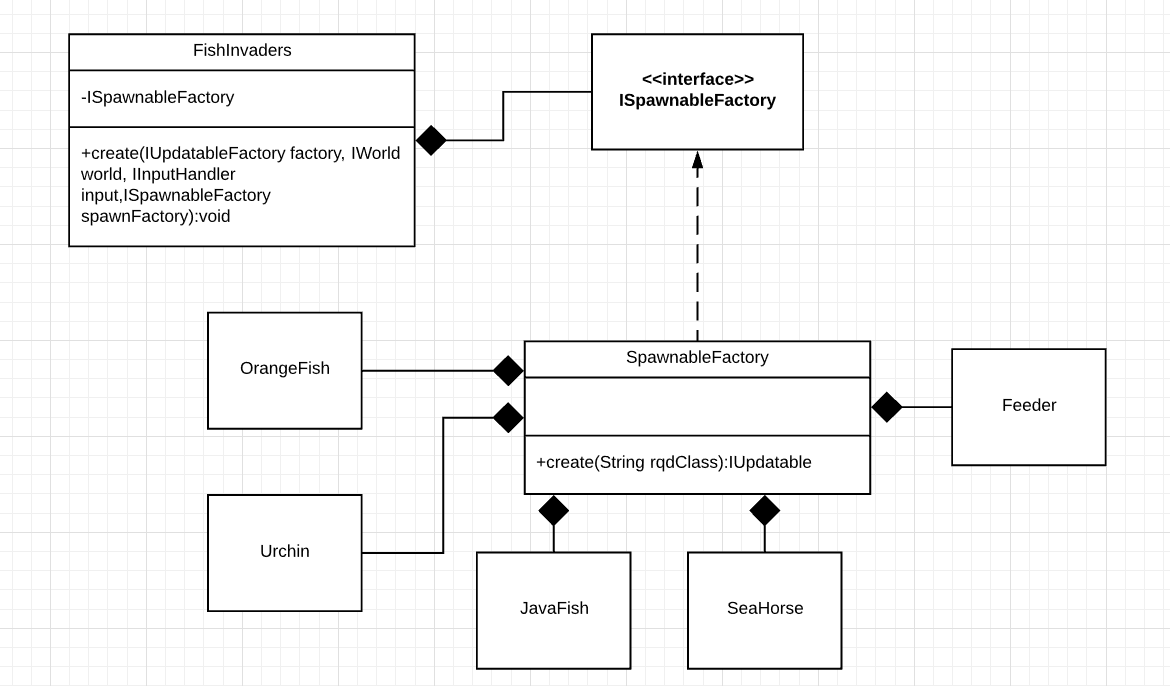
Previously the class structure for the creation of each fish looked like this. The FishInvaders had a private method called buildSpawnable and this took a class through the parameter and created an IUpdatable with the IUpdatableFactory. The problem with this was that the FishInvaders class would have to know about the class it is creating. 

This creates an unwanted dependency between the two classes. To remove this dependency, I created a Spawnable Factory, which would take in a string through the parameter and check which class it should return, depending on the string that is passed.

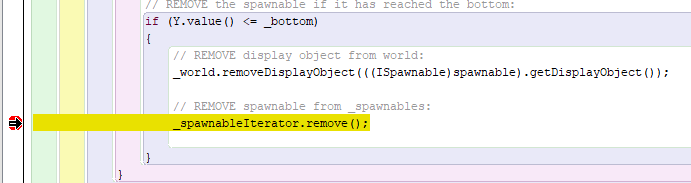




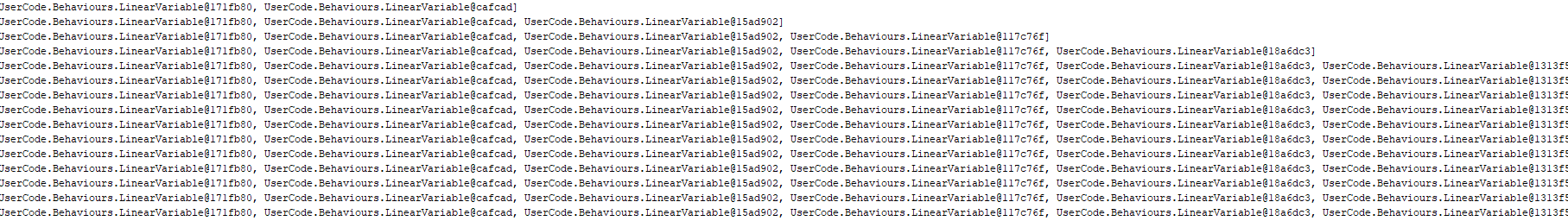
So now the FishInvaders class does not have a dependency to the fish classes or the feeder class and the FishInvader is depending on abstractions.



4.There is a bug in the software that causes spawned fish food to behave incorrectly – each piece of fish food should be spawned by left-clicking the mouse and then fall towards the bottom of the aquarium, disappearing when it reaches a point close to the bottom. Diagnose the bug and implement a change to the code that resolves it. Please log the diagnostics process that you have used.



When debugging the issue with the fish food, I found the first time the fish food is spawned and reaches the bottom, the food is removed from the iterator and display objects as expected. However, after trying this again, the food is removed before it even appears on the screen. This means that it jumps straight to the line of code where the breakpoint is. From this information I can deduce that the y value is always less than the bottom value after the first time it is removed.



The next step I took was using a system.out command to display the contents of the Ys list. It was after this that I noticed that the Y variables were not being removed from the list after the food hits the ground. This was causing the Y value of the next food that is created to be the value of the last food that was removed. To overcome this error, I just had to remove the y of the food that is being removed from the \_Ys list.



Now the code works as expected and the fish food can be infinitely spawned.

References

GeeksforGeeks. n.d. *Cohesion In Java - Geeksforgeeks*. [online] Available at: <https://www.geeksforgeeks.org/cohesion-in-java/> [Accessed 4 May 2020].

Janssen, T., 2018. *SOLID Design Principles Explained: Interface Segregation With Code Examples*. [online] Stackify. Available at: <https://stackify.com/interface-segregation-principle/> [Accessed 4 May 2020].