Object Oriented Analysis and Design

PGA Tour Superstore Simulation +

Aiden Colley, Parth Shrestha, Rayner Susanto

Youtube Demo

https://www.youtube.com/watch?v=sSkEmAg4yas

OOAD Process Statement

Personally for us, we found that most of the fundamental Object Oriented Pattern Design was useful here. In addition, the idea of creating these compartments individually and making sure they aren't tightly knit also came in handy, especially when we're doing work at different parts. Since each of our own features can "live as their own", it allowed us to develop code without stepping on each other. In addition, if we do see there's potential to scale the project, I think it'd be much easier to do it since we leave some features that are still open to changes and capable of being utilized with other compartments of code.

One issue we did face however was making sure that we're not building code for the sake of using Design Patterns rather, the other way around. We think that most of the time, we tried to make sure that there's a design pattern somewhere in our idea so it'd be much easier for us to create the code in the future. Yet, this let us to further confuse ourselves since we made unnecessary features that just seem less useful than needed. We shifted our mindset to focus more towards developing a useful application and as it turned out, we realized that it was much better for us to find patterns that can be utilized.

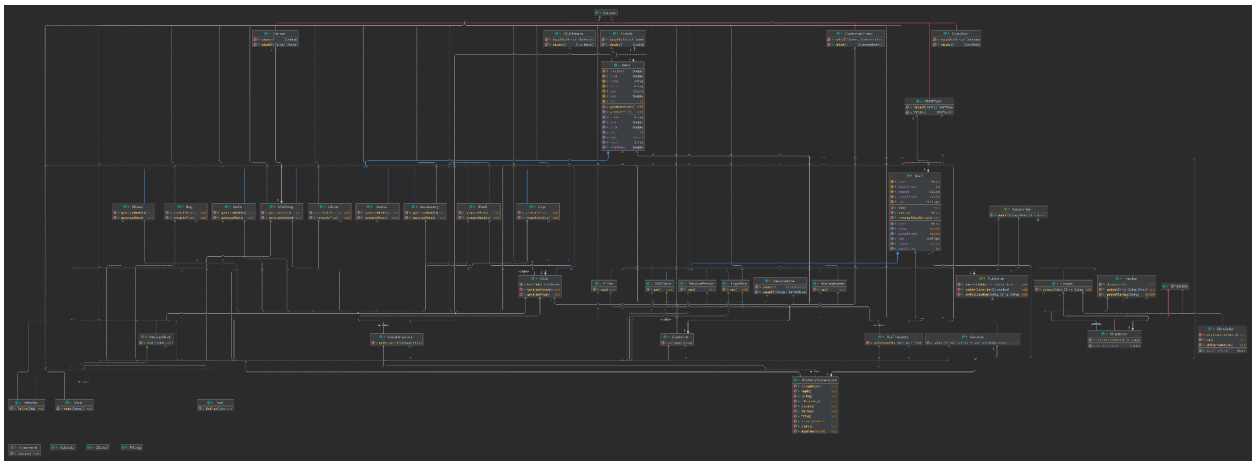
Lastly, we did enjoy making multiple changes to our project with different phases. One major learning point that we got is that code can easily change overtime. We realized that even with a fixed plan and detailed initial ideas, we would end up changing our approach since we haven't felt the problem directly. This mindset helped us to accept required changes that could ease the programming process or make the code more efficient.

Final System Statement

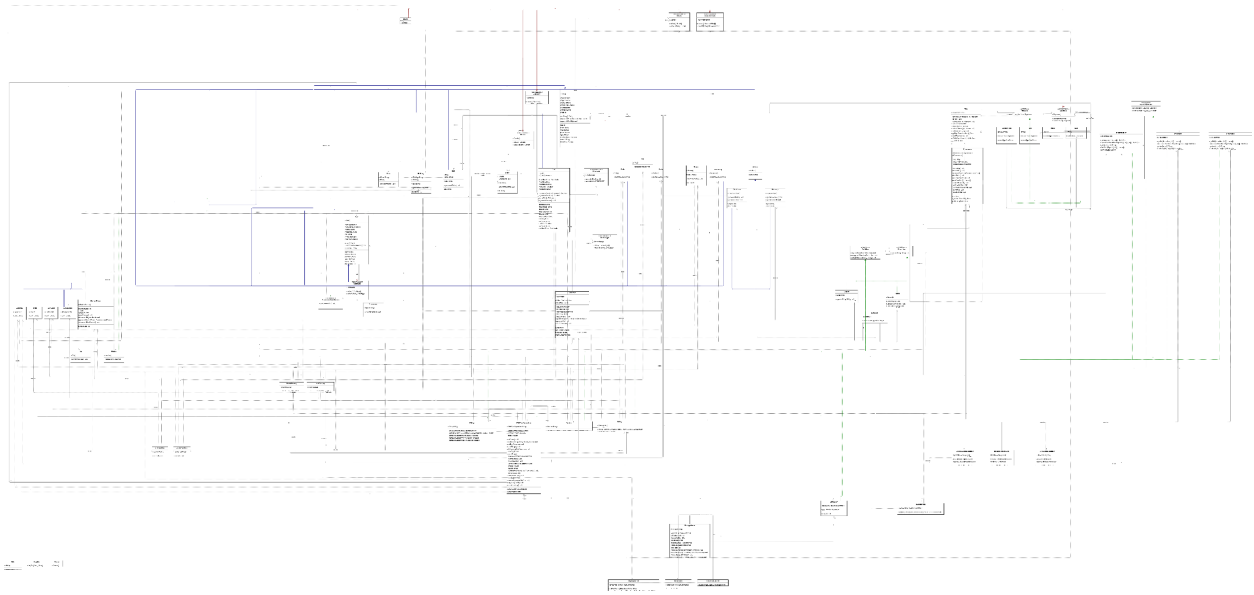
After completing this project, most of the main features that we intended to have are added here. We initially wanted to have 3 main components to this project; a simulation, a “manager mode”, and a virtual map that allows pathfinding for each item. All 3 main components are fully working. For the virtual map, initially we wanted the code to find the best path using 3 different search algorithms and just show one path. Though, due to it being potentially computation heavy, we’ve decided to go against that but instead give various path options that the user can interact with and view. We initially had the idea that this path function would be able to also sell like a simulation but it would be more beneficial for the user to know what they could potentially pick up rather than directly having the opportunity to change inventory status. As for the simulation, since our case is similar to the previous project, we decided to add a bit more to the codebase. Rather than only simple customer service, we’ve added functionality to customize the clubs into various types using fitters. Lastly, we initially wanted to implement just a simple CLI and Command Patterns to create Manager Mode. When we realized that JavaFX is a MVC Pattern itself, we decided to move forward with the UI and was able to implement that to interact with the backend, similar to a Command Pattern. The Manager Mode UI itself is slightly different to what we pictured initially. The initial idea was to have a whole dashboard that displays information for the current status of the store, we instead went ahead and created tabs for each of these information so it would be more focused.

Final Class Diagram and Comparison

Project 5/6 Diagram:



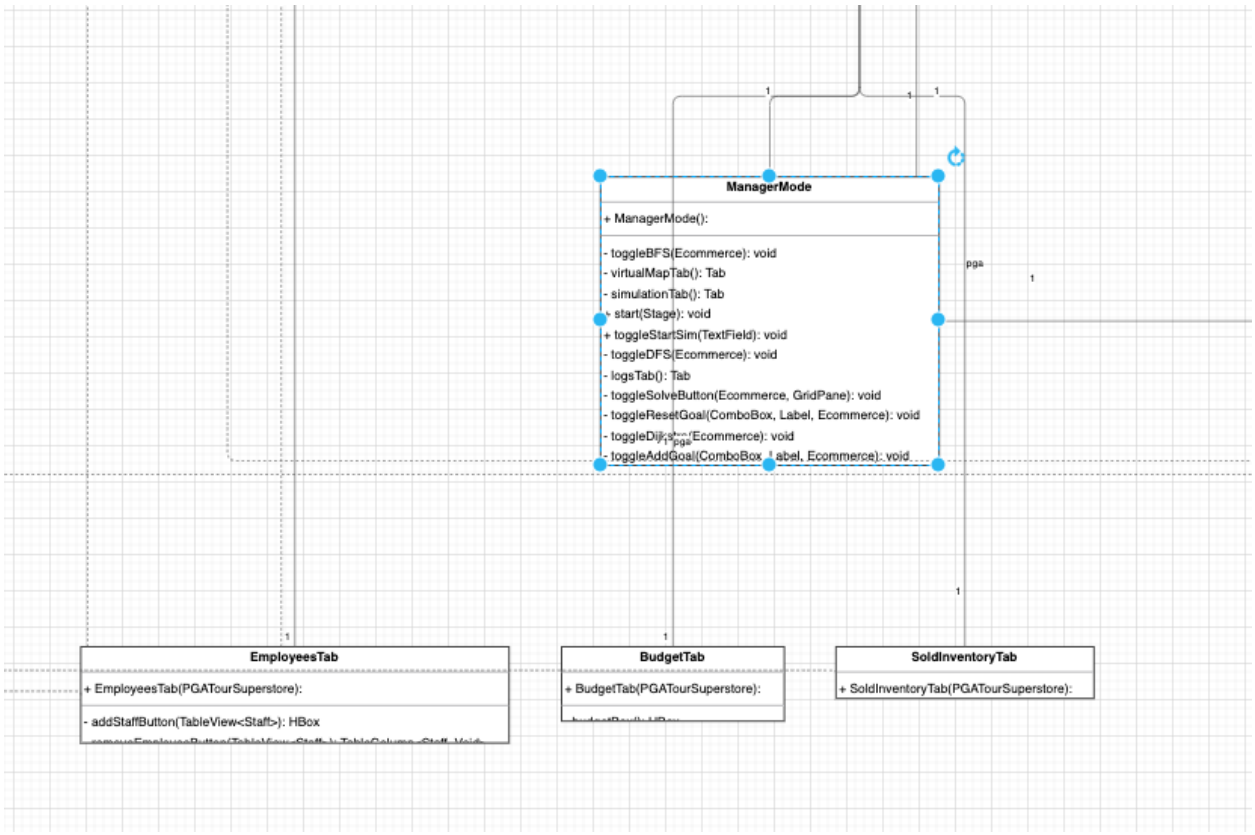
Latest Diagram



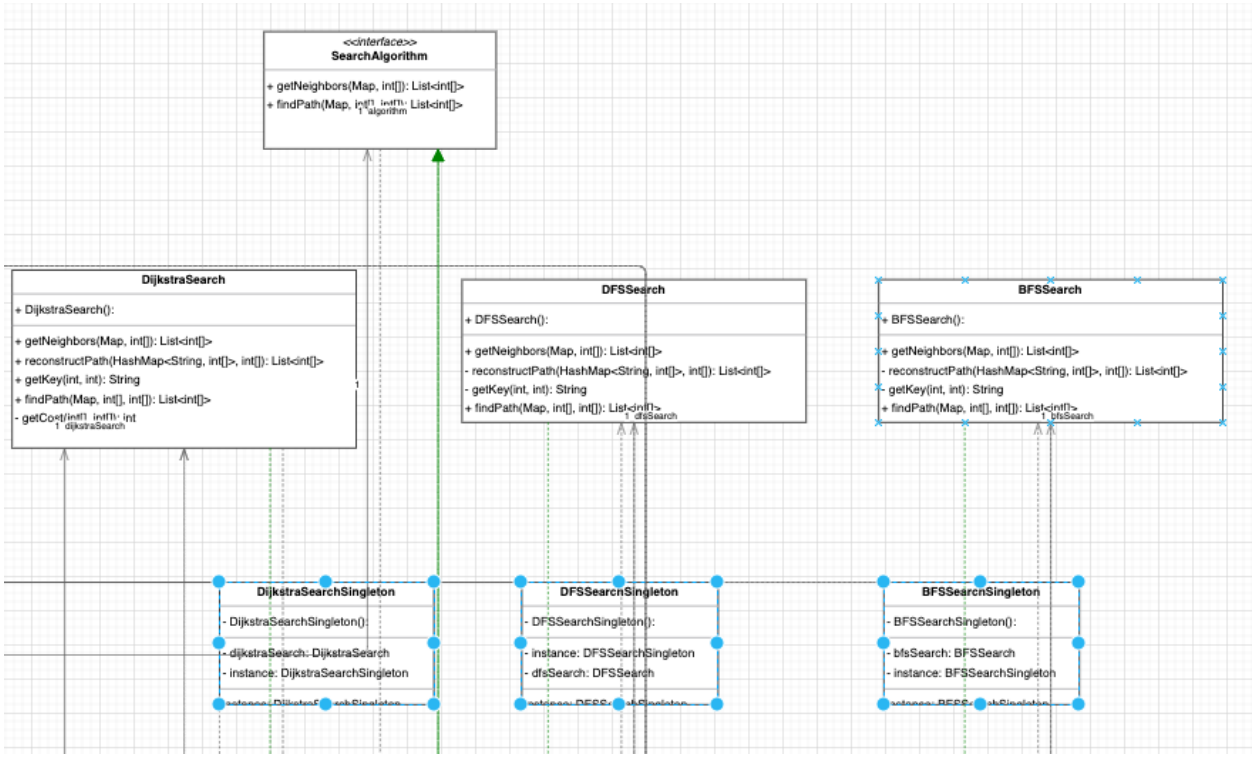
https://drive.google.com/file/d/1Wd2fjTjWcaAY26LmklmZSFLbXxSZQvTu/view?usp=sharing

(click on open with Diagrams.net or download the UML as draw.io)

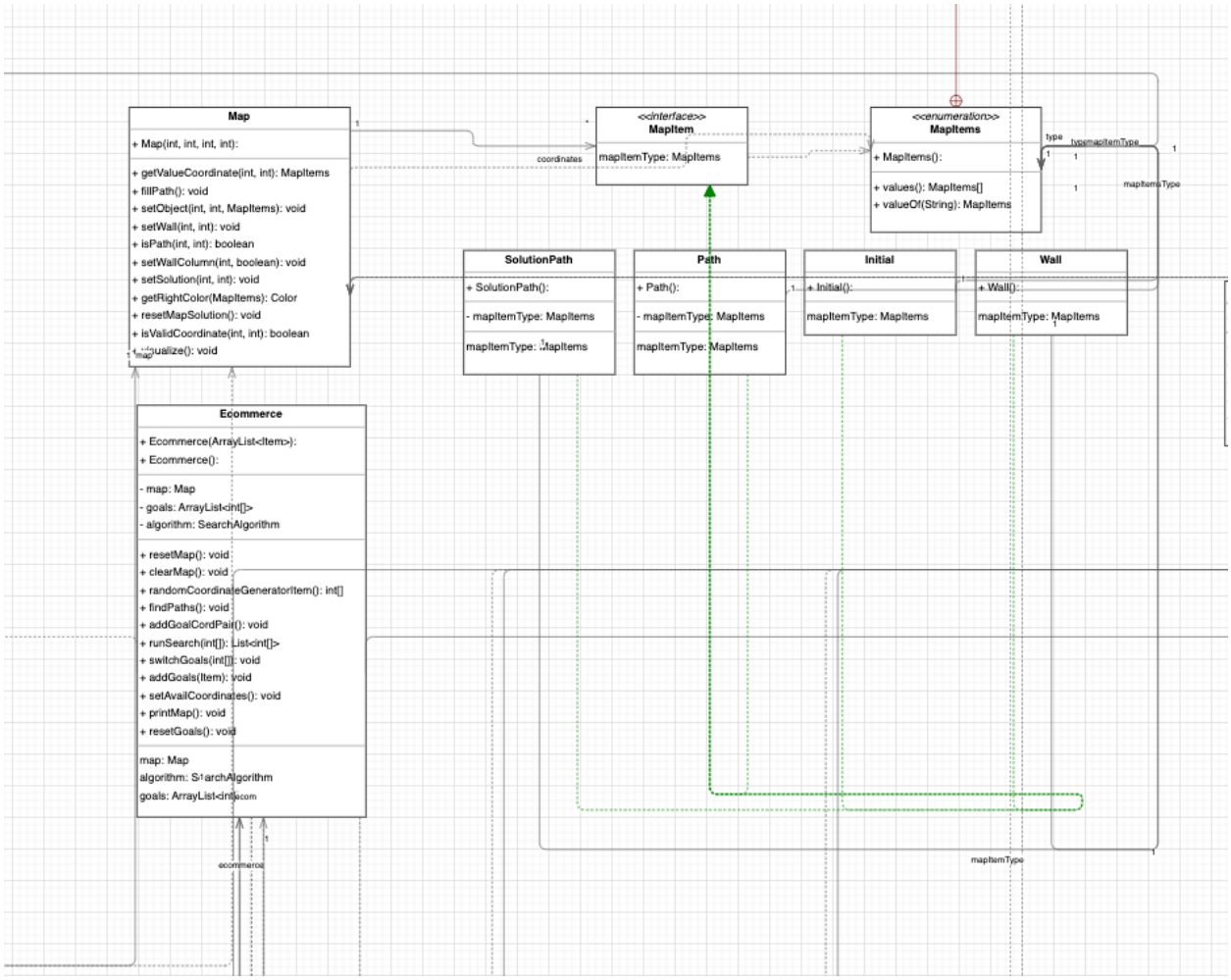
The image above doesn't do any justice to what we're trying to explain about the changes. So we will be showing individual changes with screenshots here with explanation below them.



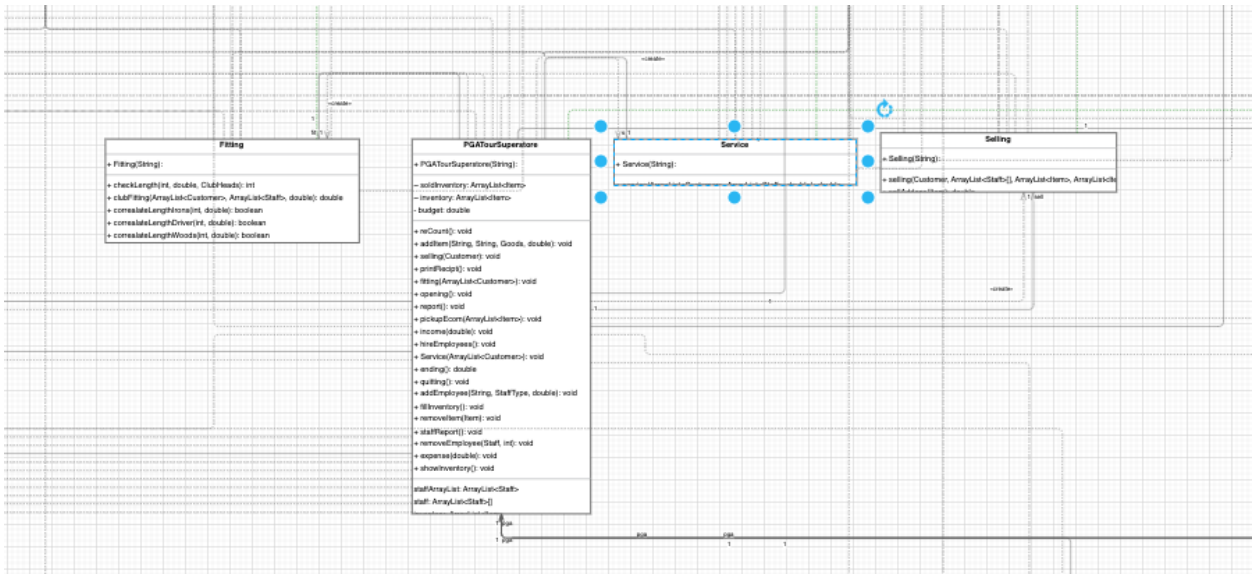
One major change that we had was how the UI was supposed to be implemented initially was how the UI would look like. We initially thought of using the Command Pattern to control the "Manager Mode" and display those changes through the CLI (similar to FNCD). We decided to use JavaFX as our tool to develop the UI which in turn creates a MVC Pattern. The View being the Desktop Application that the user sees, the Control being the buttons that are implemented in the screen and that can change the Store status, Model being the store and the virtual maps that can be used to grab items.



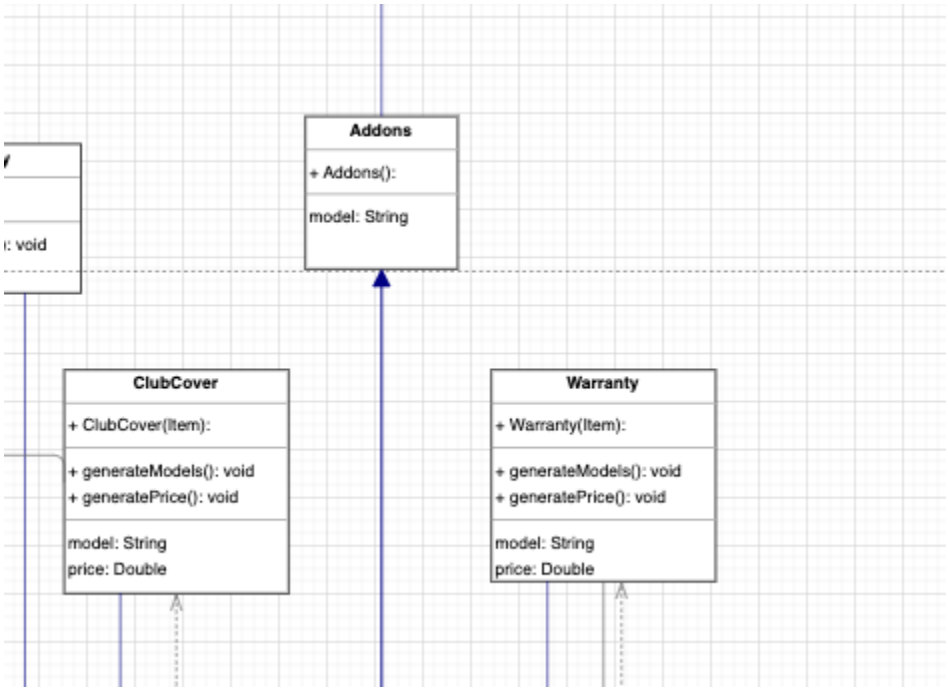
A minor change is how we initialized searchAlgorithm. We initially had the idea of implementing searchAlgorithm individual classes that have no way relating to each other since they are different. We also got from resources that most of the search algorithms rely on individual graphs that are made for them which in turn makes them produce their own results. We decided to create a singular Map class (which we kind of regretted since it's similar to Map<k,v> class from Java, in turn causing a lot of importing problems) and made sure that one Map class is being modified with each of these search algorithms. This resulted in us using a strategy pattern which we proposed in Project 6. Then an improvement is to a Singleton pattern to create these search algorithms since there's nothing that is required to be saved as a state for each searchClass and creating only one would save memory usage. In addition, the initial idea we had was to make sure that these searchAlgorithms aren't switchable with the user since we would want to compare each of those paths and pick whichever one is the most efficient, yet due to slower run time, we decided to go against that idea.

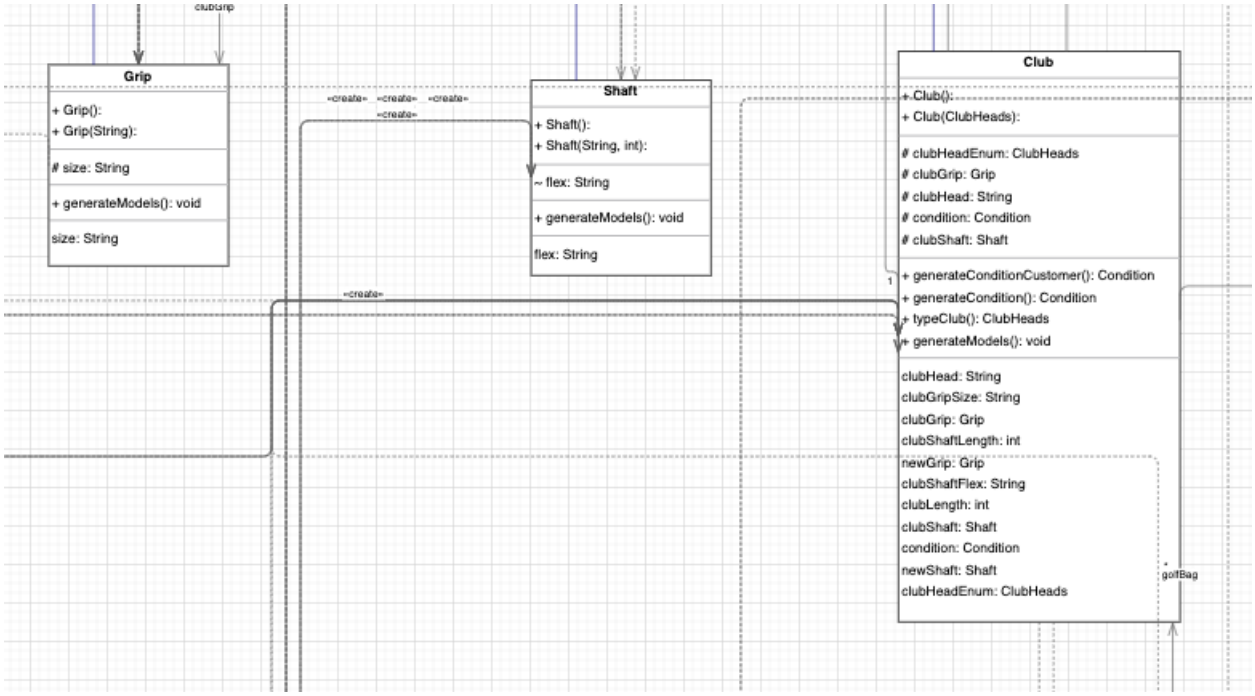


Then as for the map itself, we initially wanted to make the map inside PGA Tour since it's technically part of the store itself. Yet we realized that by doing so, the map would be tightly knit with the store that it will be hard for it to be developed in the future. So as an alternate approach, we made our own Map class and directory with all its "map item" by itself. The concept of the map item is also new compared to our project 5. We initially thought that by deriving a 2-dimensional array with integers inside would be sufficient since we wouldn't want to over complicate the project. Yet when we started implementing the idea itself, it seemed to make our code even worse since it would lead us to use arbitrary integers to claim one node is what. In addition, we wouldn't be able to identify how many items are there in each node in the future. We decided to follow a similar approach as third-party code and create individual nodes that represent something in the map (ITEMS, WALL, PATH, SOLUTION PATH).



As for the activities itself, we initially considered to try an approach where an activity would be in the same class as the store. Then again, this caused a lot of problems during our development since when we were contributing for different activities, we kept stepping on each others foot and changing other codes that weren't meant to change. This resulted in us making the activity as their own classes and making sure they return results or values that would affect the budget of the store.





This was another major change to our code on how we wanted to implement refitting, servicing and adding extras to clubs. We learned through online research and the store itself that the activities to refit and service is very common in the golfing industry. We initially thought to only change the descriptions of the golf clubs using just simple strings to

represent changes. We decided to go against that and implemented a decorator and strategy pattern to represent those activities. When a golf club is going through a certain change, we make sure to change their grip into a different grip according to their sizes and make sure that when we do service, it represents the level of experience that certain employees have.

Third Party Code Statement

As we said in our Final System Statement, our simulation here is very similar to our code from the previous project. With that, we would claim it as a third-party statement since we moved most of the code that we created from the previous project into our codebase here. Yet for some of the functionality such as Fitting and Service activity are completely new and original. This is because most of the process of Fitting and Service aren't similar to what we had initially from our previous project. We also added our own classes of Items and various Enums to make sure that it suits our project case.

Then for using JavaFX, we started off using a tutorial to setup our code with JavaFX and learn some of the simple components through here.

https://m.youtube.com/watch?v=Ope4icw6bVk

Then moving on to some of the other complicated components that weren't really touched upon the video, we consulted these websites to make our component work.

https://www.geeksforgeeks.org/javafx-vbox-class/

https://www.geeksforgeeks.org/javafx-hbox-class/

https://www.geeksforgeeks.org/javafx-combobox-with-examples/

https://www.tutorialspoint.com/javafx/layout_gridpane.htm

https://stackoverflow.com/questions/65119396/javafx-how-to-make-a-button-to-display-and-hide-label

https://www.geeksforgeeks.org/javafx-label/

https://jenkov.com/tutorials/javafx/tableview.html

Through these tutorials, we were able to learn more about these basic components and made sure to tailor them accordingly to what we need for our project.

For the Map, the idea was based on the website below:

https://www.geeksforgeeks.org/searching-algorithms-for-a-2d-arrays-matrix/

Then for the MapItems part of the code where it would be able to identify code based ITEM, WALLS, etc is the original part of the code. Then for how to display codes using System.out.print initially was based on this website:

https://www.geeksforgeeks.org/print-2-d-array-matrix-java/

Although the part where it would be visualized into the GUI (JavaFX) uses a similar concept but an original part of the code.

Then for the SearchAlgorithm, most of the code is based on these websites

https://www.baeldung.com/java-dijkstra

https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-in-java-using-priorityqueue/

https://www.baeldung.com/java-depth-first-search#:~:text=Depth%2Dfirst%20search%20(DFS),moving%20to%20explore%20another%20branch.

https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

https://favtutor.com/blogs/breadth-first-search-java

Though these algorithms are only suitable if the next node is a possible node is a “goal”. In this case, the originality of the code came when getting the neighbor from coordinate. It’s able to determine if the neighbor is a goal or just simply a WALL or PATH. As for making it into an interface for 3 different search algorithms was based on the idea of trying to implement a strategy pattern.