Ryan Whitmer

Fantasy Football Visualizer

Description: The Fantasy Football Visualizer is a java app that is used for calculating, displaying, and predicting statistics for football players. This app takes in as input the previous 4 inputs of touchdowns, touches, and yards. It also takes in the upcoming week of the football season, player name, player position, and player team. Once all the fields are filled out, the visualizer will calculate the fantasy points per week by that player (which is different by position) and show them on one graph. It will also calculate the reliability of a player given the inputs and put those on a second graph. The third and final graph is simply yards which is given in the user input. For all three of these graphs, the visualizer will show four previous weeks' worth of production as well as predicting a likely value for the upcoming/current week and displaying that as well.

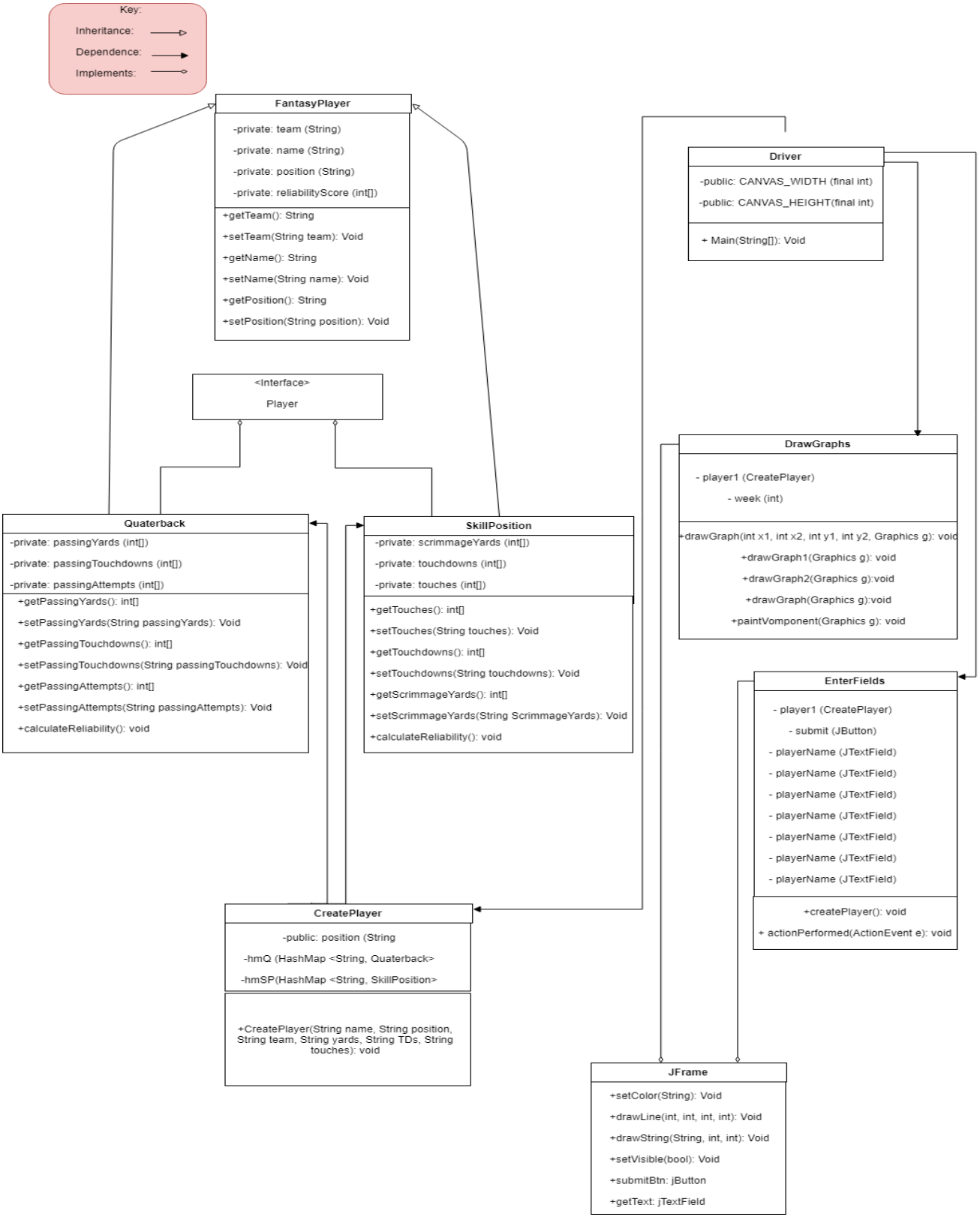
Implemented Features:

ID	Feature Name
1	Open Java App (with frontend)
2	Indicate week of the season
3	Take input for Player
4	Create a new Player
5	Display Player Name and Team
6	Generate graph of yardage
7	Calculate and Generate graph of fantasy points
8	Calculate and Generate a graph of Reliability scores
9	Create and display predictions for all 3 visualized fields

Unimplemented Features:

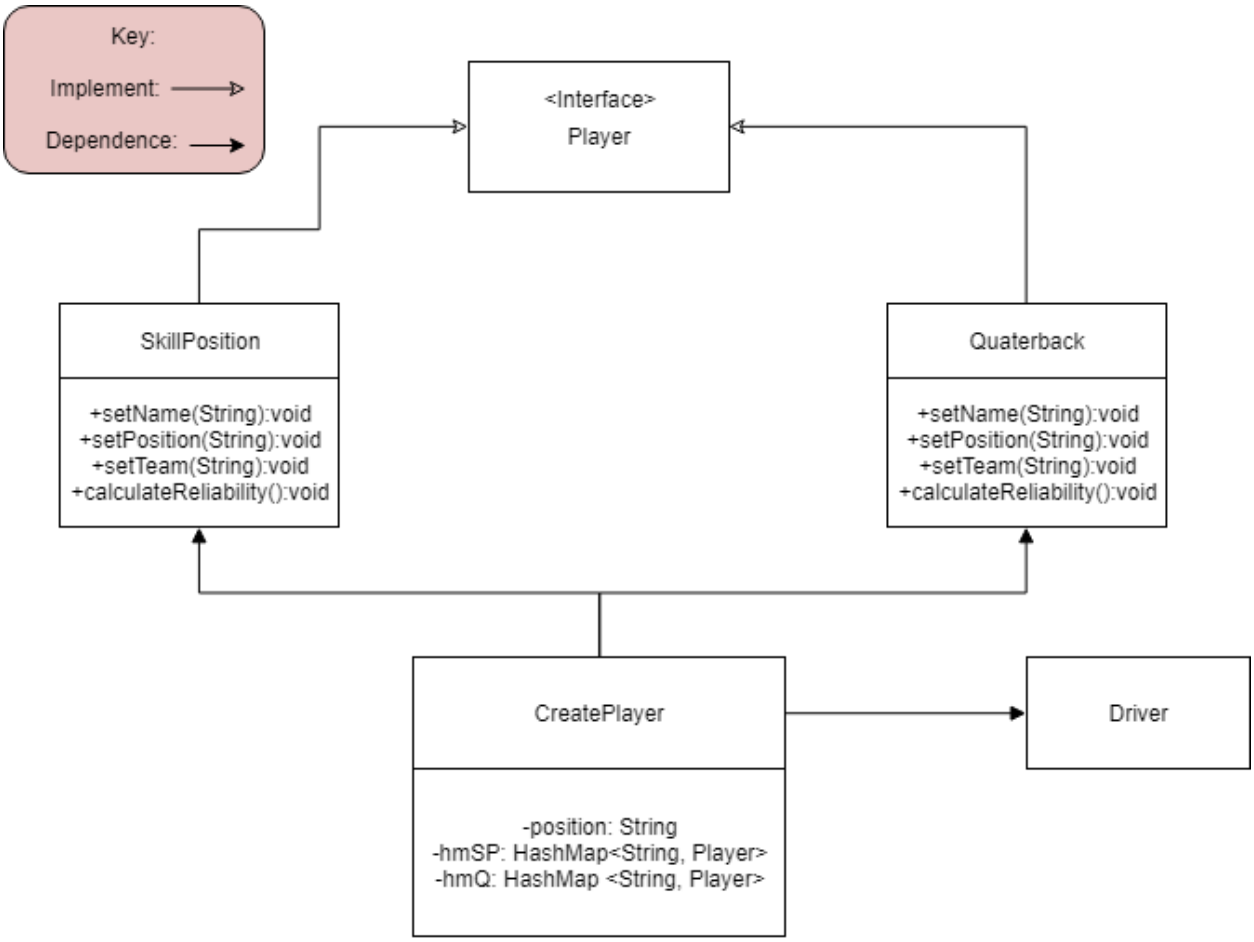
ID	Feature Name
1	Create multiple players for adjacent comparison
2	Change colors based on team input
3	Change fantasy football production values based on scoring

Final Class Diagram:



My final class diagram did change a bit from my initial diagram. Firstly, the way CreatePlayer is implemented has changed a lot. When FlyWeight became part of my project, the drive only needed to care about CreatePlayer and no other back-end. This also meant that it depended on both Quaterback as well as SkillPosition, but nothing else. FlyWeight also meant adding an interface that I created that was implemented by both Quaterback and SkillPosition. Finally, the changes over on the right of the diagram all stemmed from doing more research on how to implement my JFrame front end and realizing I could use an extra class, and I'd need a lot of extra helper variables/functions.

Flyweight diagram:



(This class diagram is for showing flyweight interaction, not filled out entirely for present classes)

The way I implemented flyweight into my project is by sending every player creation through the 'CreatePlayer' class and storing them in a Hash Map. I also created an interface that, regardless of player position, applied to the player. This made it possible to fill in the Hash Maps regardless of which specific player a SkillPosition player was. I also would have really liked to create multiple players to compare on the resultant graphs, and I set myself up to do so very nicely, but the UI implementation for this did not get done in time. I selected this design pattern

because I found it perfect for creating and storing many objects that you don't know the exact type of. I also found it fairly intuitive and organized to implement.

What Have I learned: The main thing that I learned about process of analysis and design from this project is that projects are going to change throughout their development. This project had a lot of its initial functionalities met, but some that weren't simply ended up being a lower priority or unnecessarily difficult for their value. As a programmer I had to be very ready to adapt as this project started coming to fruition and snags were hit. This could also go the other way where throughout the project I learned a lot and could add functionalities I never thought I would've. With slightly more time, or if I were to start another similar project, it would be very feasible and easy to accomplish what I thought would've been the most difficult stretch goal. On top of all that, my initial class diagram and mental image of this final project have changed in a few different ways, and I had to be ready to make those changes and not stubbornly hang on to my initial view of this project.