Final Project Submission - Report

Smash Tournament Bracket Generator
Katariya Edfors and Athena Parker

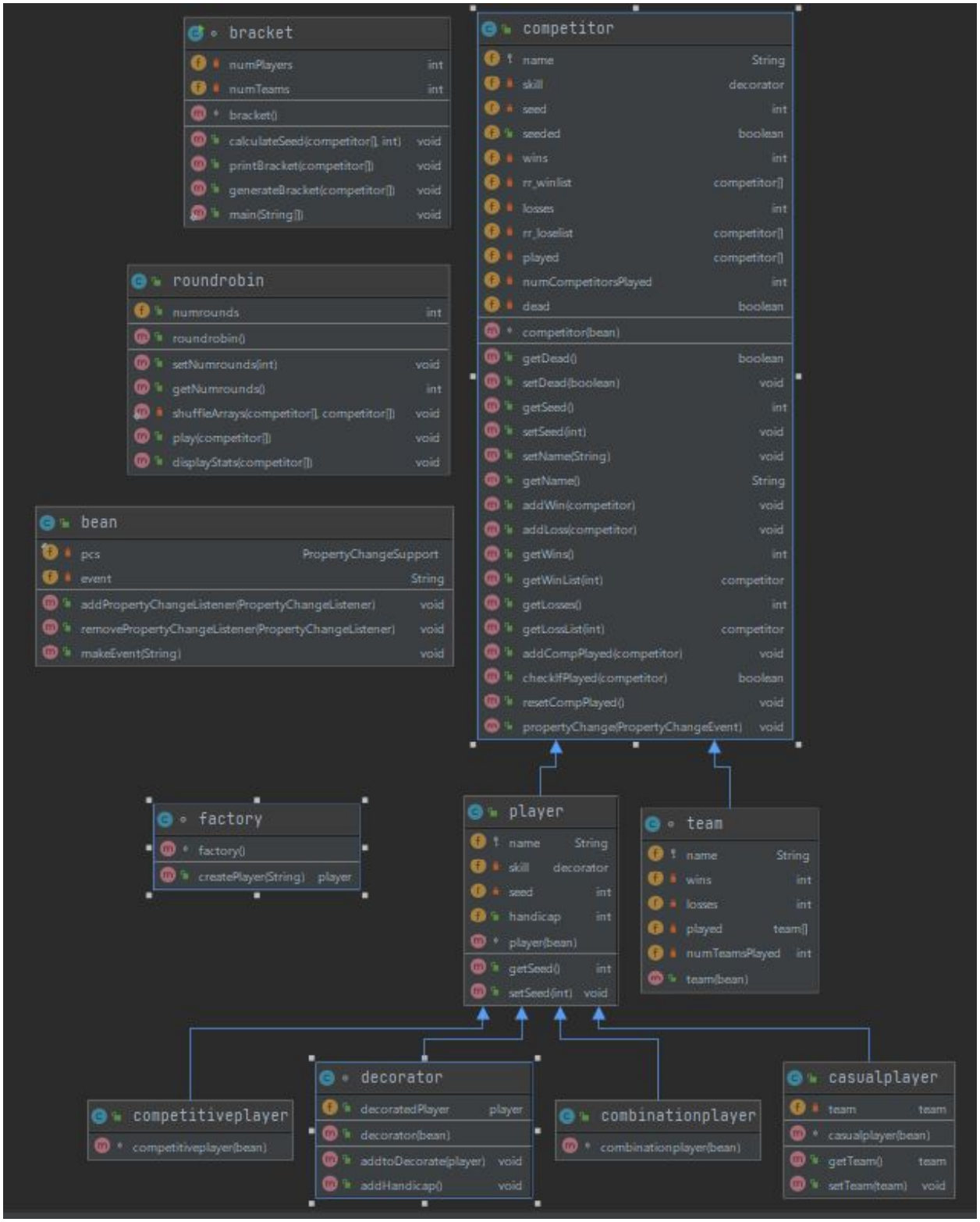
Final State of System Statement

Our project implements the Observer Pattern, Factory Pattern, and Decorator Pattern. We first read in the player information from an Excel sheet, populate an array of Competitor objects, and create random pairings in a pre-bracket Round Robin. We then keep track of wins and losses entered onto the Excel sheet by the tournament organizers, and using a combination of each player's win ratio as well as the win ratios of the people they have played against, we calculate and display their seed in the bracket.

We did implement almost all the main functionalities planned, though a couple of side projects (smashdown functionality from project 4 and larger team functionality from project 5) did not get implemented in the end. Also, we generated the bracket in the terminal via user inputs for the winner of each matchup as the formatting of the .csv file would be corrupted when shifting from excel to java in the upload. With each "winner" input, the bracket adapts the next round with the remaining players.

Final Class Diagram and Comparison Statement

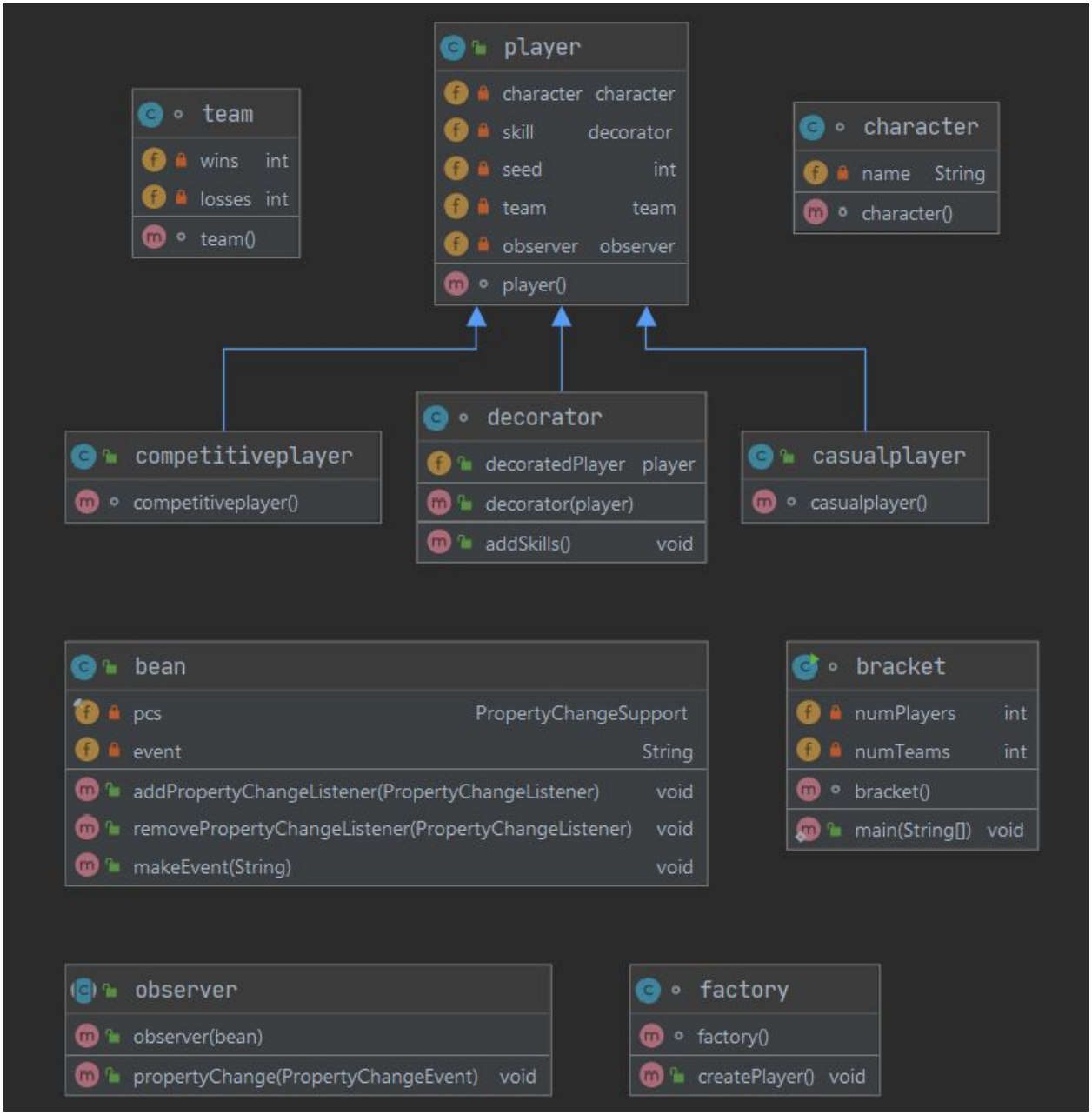
Project 6 UML:



Patterns (selected):

Factory Pattern - creates competitors
Decorator Pattern - adds a handicap to players who are too advanced for the competitor pool
Observer Pattern (implemented in Competitor) - all competitors observe the bracket, and display seed information when round robin has completed

Project 4 UML:



The major changes that have taken place is that both Player and Team now inherit from a class Competitor, and instead of being its own class, Observer is now implemented in Competitor. We decided rather than have an observer for each competitor, we could simplify it by just having competitors themselves be the observers. We have also removed the Character object as whileit is a fun feature for a casual game between friends, the attribute isn't needed for an organized tournament. We have added a round robin object to run a round robin game, as well as export all match ups to an Excel sheet, and also added Excel file input and seed calculations to our main bracket object.

Third Party Code vs. Original Code Statement

All code in our project is original with the exception of the shuffle array action in the RoundRobin class. The shuffle used in our project is adapted from a Fisher-Yates Shuffle implementation from https://stackoverflow.com/questions/1519736/random-shuffling-of-an-array and a reference is made to the source in our comments under our adapted implementation.

Statement on the OOAD Process for Overall Semester Project

One design process issue we faced was figuring out how to best implement our patterns into our project. We went through several iterations on how our Observer and Decorator patterns specifically could be most useful in our system, and we are now happy with our result.

A key design process issue that significantly slowed our progress was implementing a user-friendly excel output/input so that the user could use the project without a strong knowledge of java. Unfortunately when trying to install and use an excel api, like jxl, we could not get the api functioning for our IntelliJ IDE. Switching between Excel and .csv formatting for Java to intake also proved difficult, as the formatting would be corrupted in downloading the excel info as a .csv and would alter the accuracy when reading in the data. Ultimately, our steps to adapt for the reformatting were unsuccessful and we resolved to take the final bracket-generator data via terminal inputs with user prompts.

Another design process issue was with modification of objects in Java. We had been under the impression that a copy of an object will not modify the original object, and it had caused quite a bit of confusion.

A final design process element we had kept in mind was how best to implement inheritance. Originally, we had players and teams as separate entities. As we worked through the project, we realized that their functionalities are very similar, so we then created a class Competitor that both Player and Team inherit from.