

# Driver & Interface Adapter

# AppState

- Serializable Class that will store all of the current matches members
- Can be loaded and unloaded to save and reload a fantasy league's data

Parent: None

Subclasses: None

Layer: Interface Adapters

Relationships:

- SportsApp

# DataContainer (Interface)

- DataContainer's subclasses will contain the data on players and teams
- When the app starts, data container subclass will be empty. Then, when a command is ran, it will check if it has the appropriate data, if not, it will load it in and save it, and if it does, it will just pass it right away

Parent: None

Subclasses: CSVDataContainer

Layer: Interface Adapters

Relationships:

- SportsApp
- CSVDataContainer

# CSVDataContainer

- Loads the necessary data from a CSV file
- Implements the methods set by the DataContainer interfaces: getPlayer and getTeam

Parent: DataContainer

Subclasses: None

Layer: Interface Adapters

Relationships:

- DataContainer

# SportsApp

- Starting point of the app
- Prints starting instructions for the user to follow
- Accepts input from the user and passes it to the CommandManager
- Prints out the output received from the CommandManager

Parent: None

Subclasses: None

Layer: Framework & Drivers

Relationships:

- CommandManager

# CommandManager

- Parses the input string into a command and its arguments
- Identifies the command and passes it to an appropriate object

Parent: None

Subclasses: None

Layer: Interface Adapter

Relationships:

- InputParser
- PlayerStatManagerFacade
- PlayerStatComparerFacade
- PlayerStatPredictorFacade
- TeamStatManager
- TeamStatComparer
- TeamStatPredictor
- LeagueMemberManager

# InputParser

- The class is responsible for parsing the user's input and extracting the arguments out of the input
- It will store the keyword command and the arguments for the command separately

Parent: None

Subclasses: None

Layer: Interface Adapter

- CommandManager

Related to Teams



# TeamStatManagerFacade

- Facade class which accepts an argument requesting a statistic, checks the sport requested, and passes the argument to the appropriate sport's StatManager

Parent: None  
Subclasses: None  
Layer: Use Case

- HockeyTeamStatManager
- *BaseballTeamStatManager (not implemented)*

# TeamStatManager (Abstract)

- Find or compute statistics about a given Team

Parent: None  
Subclasses: HockeyTeamStatManager,  
TennisTeamStatManager,  
BaseballTeamStatManager  
Layer: Use Case

- None

# HockeyTeamStatManager

- Find or compute statistics about a given HockeyTeam

Parent: TeamStatManager  
Subclasses: None  
Layer: Use Case

- TeamManager
- HockeyTeam

# BaseballTeamStatManager

- Find or compute statistics about a given BaseballTeam

Parent: TeamStatManager  
Subclasses: None  
Layer: Use Case

- TeamManager
- BaseballTeam

# Team (Abstract)

- Store a team's name and provide a template for concrete team classes
- Getters and Setters for the data

Parent: None  
Subclasses: HockeyTeam,  
*BaseballTeam*  
Layer: Entity

- HockeyTeam

# HockeyTeam

- Store the information for a hockey team
- Stores goals for, wins, losses, goals for, goals against, face off win percentage, shots for, shots against, overtime losses,
- Getters and Setters for above

Parent: Team  
Subclasses: None  
Layer: Entity

- TeamManager

# *BaseballTeam*

Parent: Team  
Subclasses: None  
Layer: Entity

- Store the information specified in team class
- Also stores games started, complete games, shutouts, saves, save opportunities, innings pitched, hits allowed, runs allowed, earned runs, home runs allowed, hit batsmen, at bats, runs, hits, doubles, triples, home runs, run batted in, walks, strikeouts, stolen bases, caught stealing
- Getters and Setters for above

- TeamManager

Related to Members & Betting



# LeagueMemberManager

- Create and record the Members in the fantasy league
- Create and record the ongoing Matches
- Notify stored Matches when a Member bets on them or when their outcome is resolved

Parent: None  
Subclasses: None  
Layer: Use Case

- Member
- Match

# LeagueMember

- Represent a Member of a fantasy league, who bets on games
- Stores the Member's name
- Tracks the amount of matches they have predicted correctly and incorrectly

Parent: None  
Subclasses: None  
Layer: Entity

- None

# Match

- Store the two teams who are competing in the match
- Getters and setters for above
- Record and store which Members have bet on which outcomes of the match
- After the match ends, update players who bet correctly and who bet incorrectly

Parent: None  
Subclasses: None  
Layer: Entity

- MemberManager

Related to Players

# PlayerStatManager

- Abstract Class which is Superclass of each sports' concrete stat manager class
- Stores player list and list of stats which can be returned.

Parent: None

Subclasses:

HockeyPlayerStatManager,  
TennisPlayerStatManager

Layer: Use Case

- None

# PlayerStatManagerFacade

- Facade class which accepts an argument requesting a statistic, checks the sport requested, and passes the argument to the appropriate sport's StatManager

Parent: None

Subclasses: None

Layer: Use Case

- HockeyPlayerStatManager
- TennisPlayerStatManager
- BaseballPlayerStatManager

# HockeyPlayerStatManager

- Return the value of a stat (or all stats), given a hockey player's name, a season name, and a stat
- Stats that can be reported are:
  - See HockeyPlayer card (any of the information being stored by HockeyPlayer can be reported)

Parent: PlayerStatManager

Subclasses: None

Layer: Use Case

- HockeyPlayer
- DataContainer

# TennisPlayerStatManager

- Return the value of a stat (or all stats), given a tennis player's name, a tournament name, and a stat
- Stats that can be reported are:
  - See TennisPlayer card (any of the information being stored by TennisPlayer can be reported)

Parent: PlayerStatManager

Subclasses: None

Layer: Use Case

- TennisPlayer
- DataContainer



# BaseballPlayerStatManager

- Return the value of a stat (or all stats), given a baseball player's name, a season, and a stat
- See BaseballPlayer card to see what stats can be reported

Parent: PlayerStatManager

Subclasses: None

Layer: Use Case

- BaseballPlayer
- DataContainer

# PlayerStatComparer

- Abstract Class which is Superclass of each sports' concrete stat comparer class
- Stores player list and list of stats which can be compared.

Parent: None

Subclasses: HockeyPlayerStatComparer,  
TennisPlayerStatComparer

Layer: Use Case

- None

# PlayerStatComparerFacade

- Facade class which accepts an argument requesting comparison of statistics, checks the sport requested, and passes the argument to the appropriate sport's StatComparer

Parent: None

Subclasses: None

Layer: Use Case

- HockeyPlayerStatComparer
- TennisPlayerStatComparer
- BaseballPlayerStatComparer

# HockeyPlayerStatComparer

- Compare two or more hockey players on a given statistic in a specific season
- Stats that can be compared:
  - number of games played
  - number of goals
  - number of assists
  - number of points
  - shooting percentage
  - number of shots

Parent: PlayerStatComparer

Subclasses: None

Layer: Use Case

- HockeyPlayer
- DataContainer

# TennisPlayerStatComparer

- Compare two tennis players who participated in a competition based on a given stat
- Stats that can be compared are:
  - number of aces
  - number of double faults
  - number of serve points
  - number of first serves
  - number of break points saved

Parent: PlayerStatComparer

Subclasses: None

Layer: Use Case

- TennisPlayer
- DataContainer

# BaseballPlayerStatComparer

- Compare multiple baseball players on a given stat for a season
- Statistics that can be compared are:
  - number of games played
  - number of at bats
  - number of runs
  - number of hits
  - number of home runs
  - number of runs batted in
  - number of strikeouts
  - average hits per bat

Parent: PlayerStatComparer

Subclasses: None

Layer: Use Case

- BaseballPlayer
- DataContainer

# PlayerStatPredictor

- Abstract Class which is Superclass of each sports' concrete stat predictor class
- Stores player list and list of stats which can be predicted.

Parent: None  
Subclasses: None  
Layer: Use Case

- None

# PlayerStatPredictorFacade

- Facade class which accepts an argument requesting prediction of statistic, checks the sport requested, and passes the argument to the appropriate sport's StatPredictor

Parent: None

Subclasses: None

Layer: Use Case

- HockeyPlayerStatPredictor
- TennisPlayerStatPredictor
- BaseballPlayerStatPredictor



# HockeyPlayerStatPredictor

- Given a hockey player's name and a stat, predict their future statistic using linear regression
- Stats that can be predicted are:
  - number of games played
  - number of goals
  - number of assists
  - number of points
  - shooting percentage
  - number of shots

Parent: PlayerStatPredictor

Subclasses: None

Layer: Use Case

- HockeyPlayer
- DataContainer

# TennisPlayerStatPredictor

- Given a tennis' player's name and stat, predict their future statistic with linear regression
- Stats that can be predicted are:
  - number of aces
  - number of double faults
  - number of serve points
  - number of first serves
  - number of break points saved

Parent: PlayerStatPredictor

Subclasses: None

Layer: Use Case

- TennisPlayer
- TennisPlayerList

# BaseballPlayerStatPredictor

- Given a baseball player's name and stat, predict their future statistic with linear regression
- Stats that can be predicted are:
  - number of games played
  - number of at bats
  - number of runs
  - number of hits
  - number of home runs
  - number of runs batted in
  - number of strikeouts
  - average hits per bat

Parent: PlayerStatPredictor

Subclasses: None

Layer: Use Case

- BaseballPlayer
- DataContainer

# Player (Abstact)

- Store player's name
- Getter and setter for above

Parent: None  
Subclasses: HockeyPlayer  
Layer: Entity

- None

# HockeyPlayer

- Store the season, position, number of games played, number of goals, number of assists, number of points, shooting percentage, number of shots, and skater shoots

Parent: HockeyPlayer

Subclasses: None

Layer: Entity

- None

# TennisPlayer

- Store a tennis player's:
  - age
  - nationality (represented by the 3 letter IOC code for their country)
  - number of aces
  - number of double faults
  - number of first serves
  - number of serve points
  - number of break points saved
- Getter and setters for above

Parent: Player  
Subclasses: None  
Layer: Entity

- None

# BaseballPlayer

- For all seasons, store a baseball player's:
  - position
  - games played
  - bats
  - runs
  - hits
  - home runs
  - runs batted in
  - strike outs
  - average hits per at bat
- Getter and setters for above

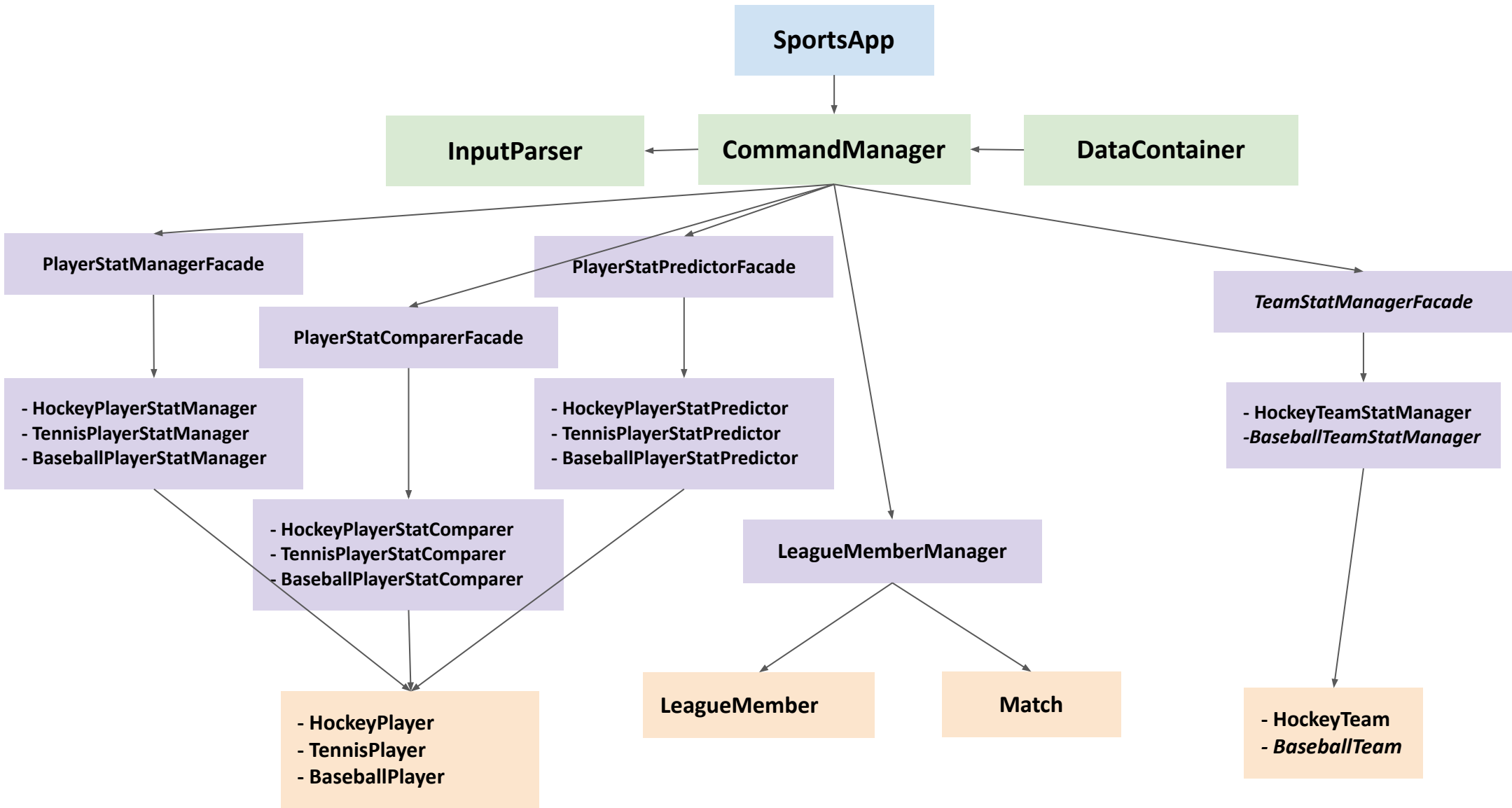
Parent: Player  
Subclasses: None  
Layer: Entity

- None

- Framework and Drivers
- Interface Adapters
- Use Cases
- Entities

## Simplified Class Diagram

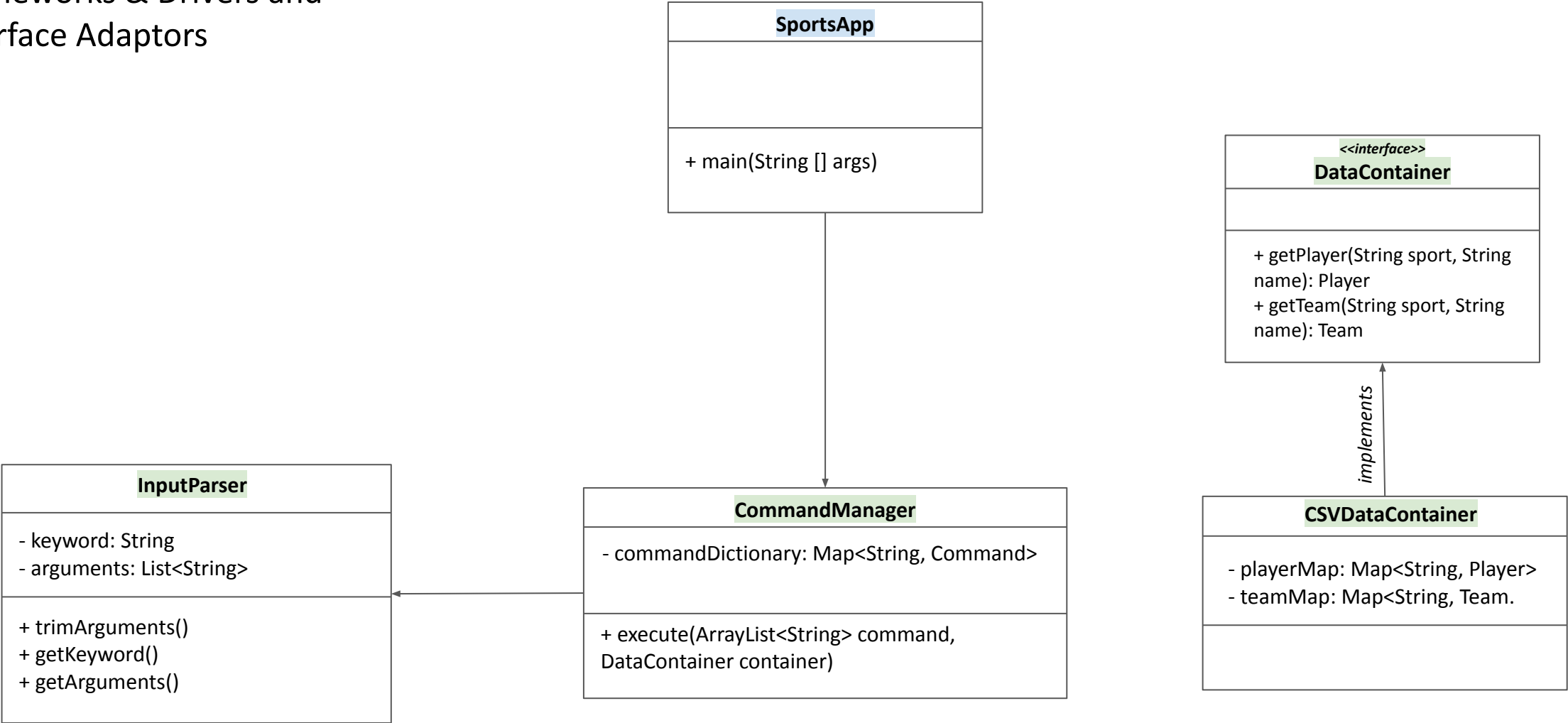




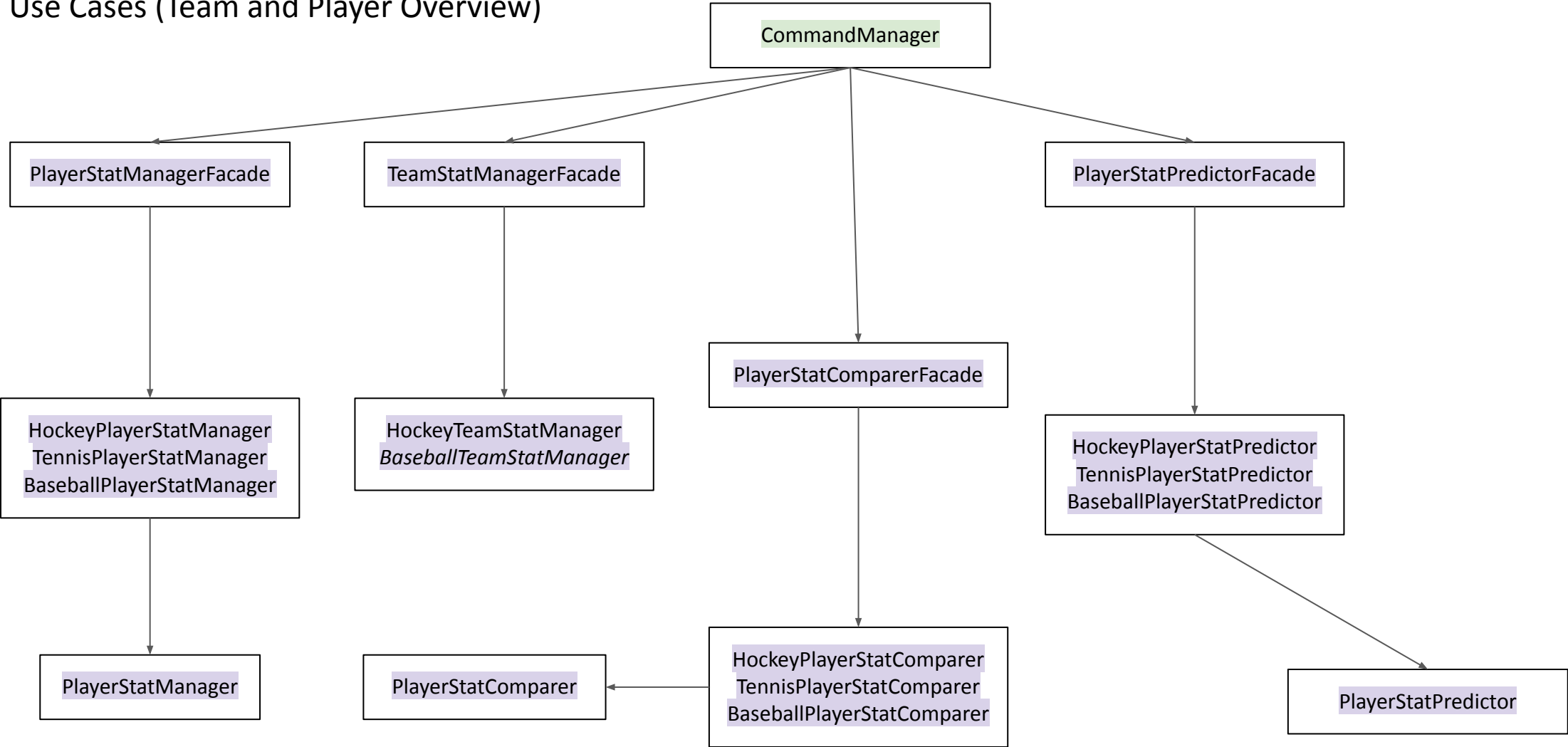
- Framework and Drivers
- Interface Adapters
- Use Cases
- Entities

## **Class Diagrams by Clean Architecture Layer**

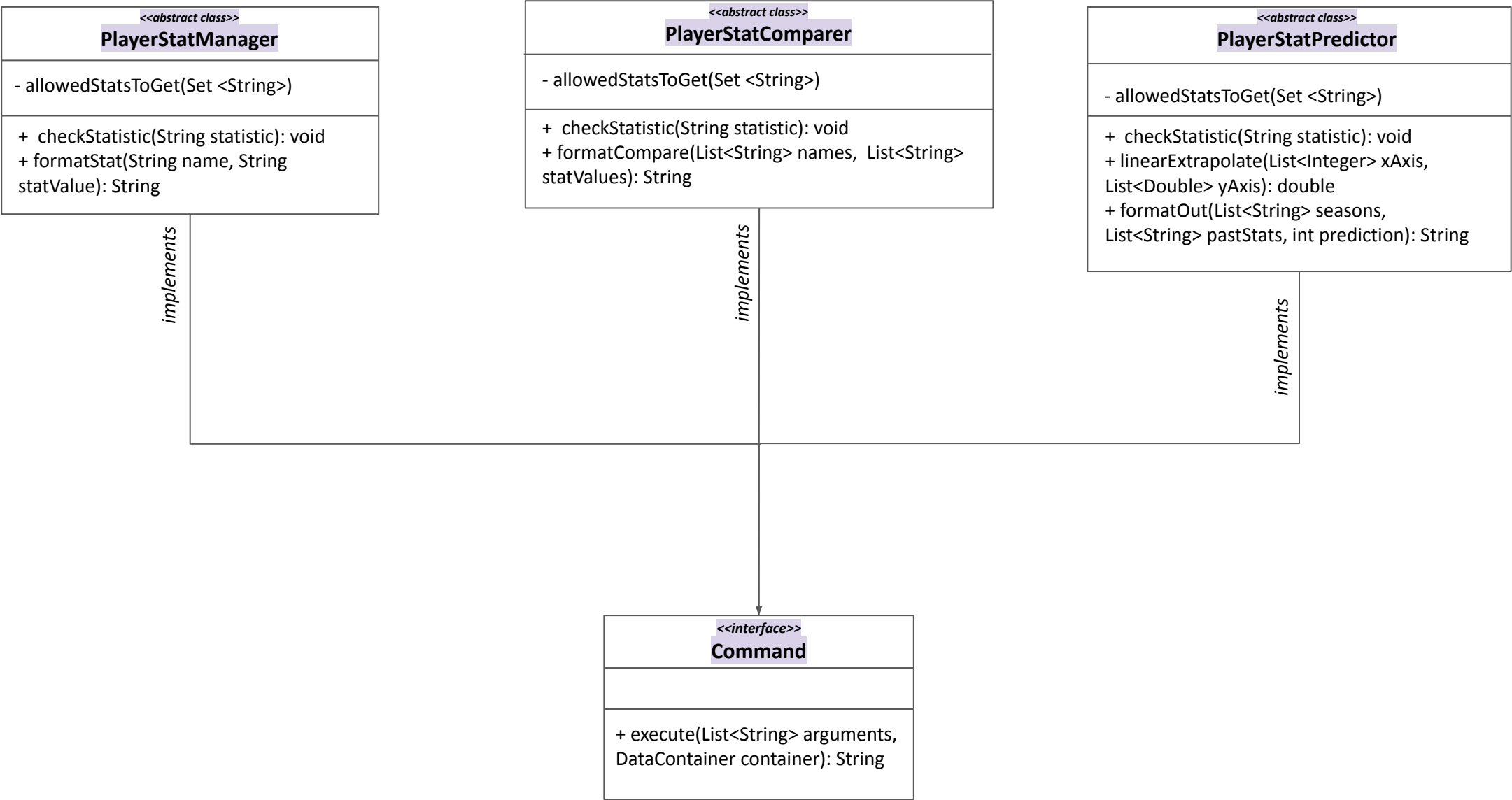
# Frameworks & Drivers and Interface Adaptors



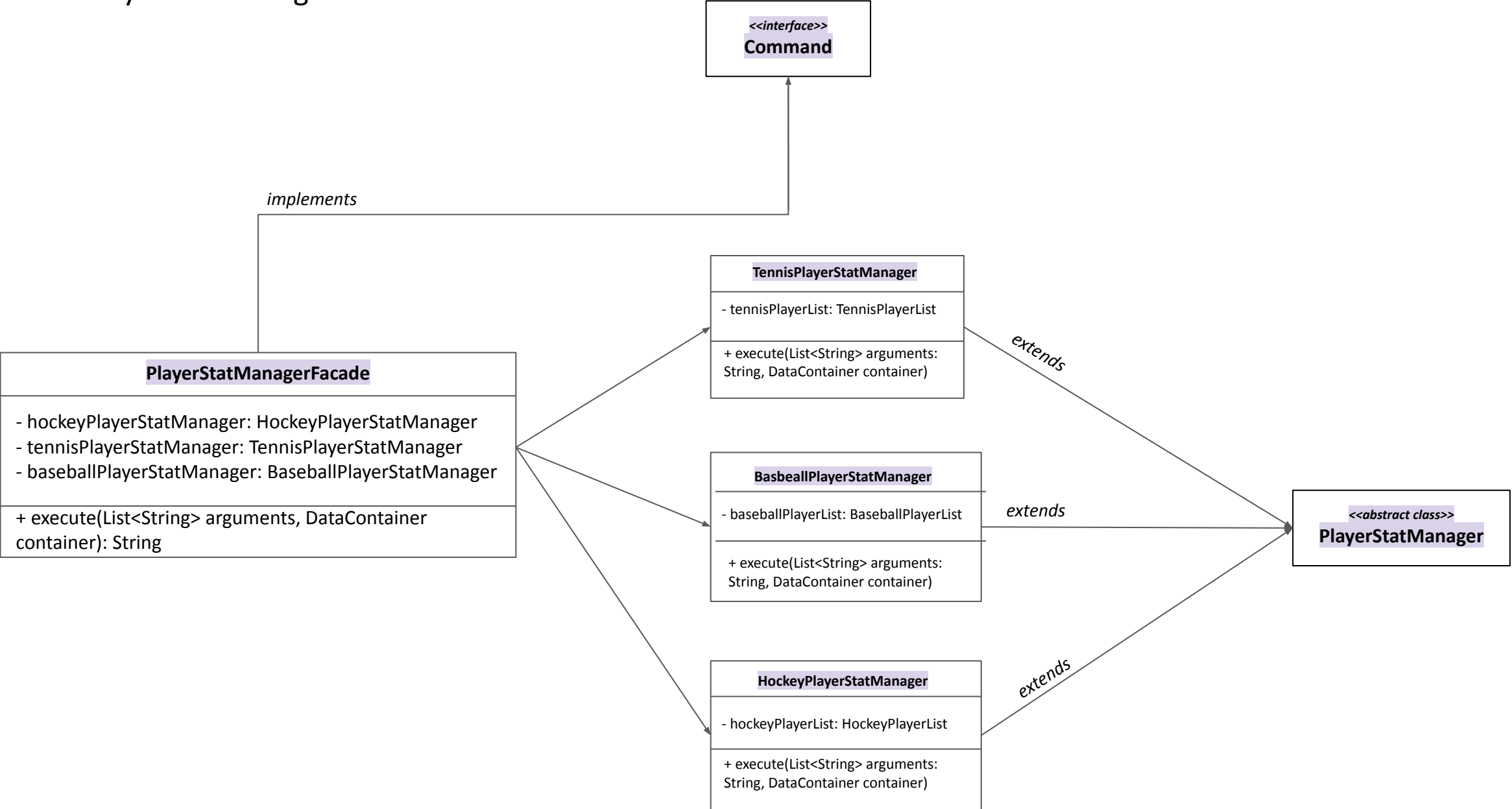
Use Cases (Team and Player Overview)



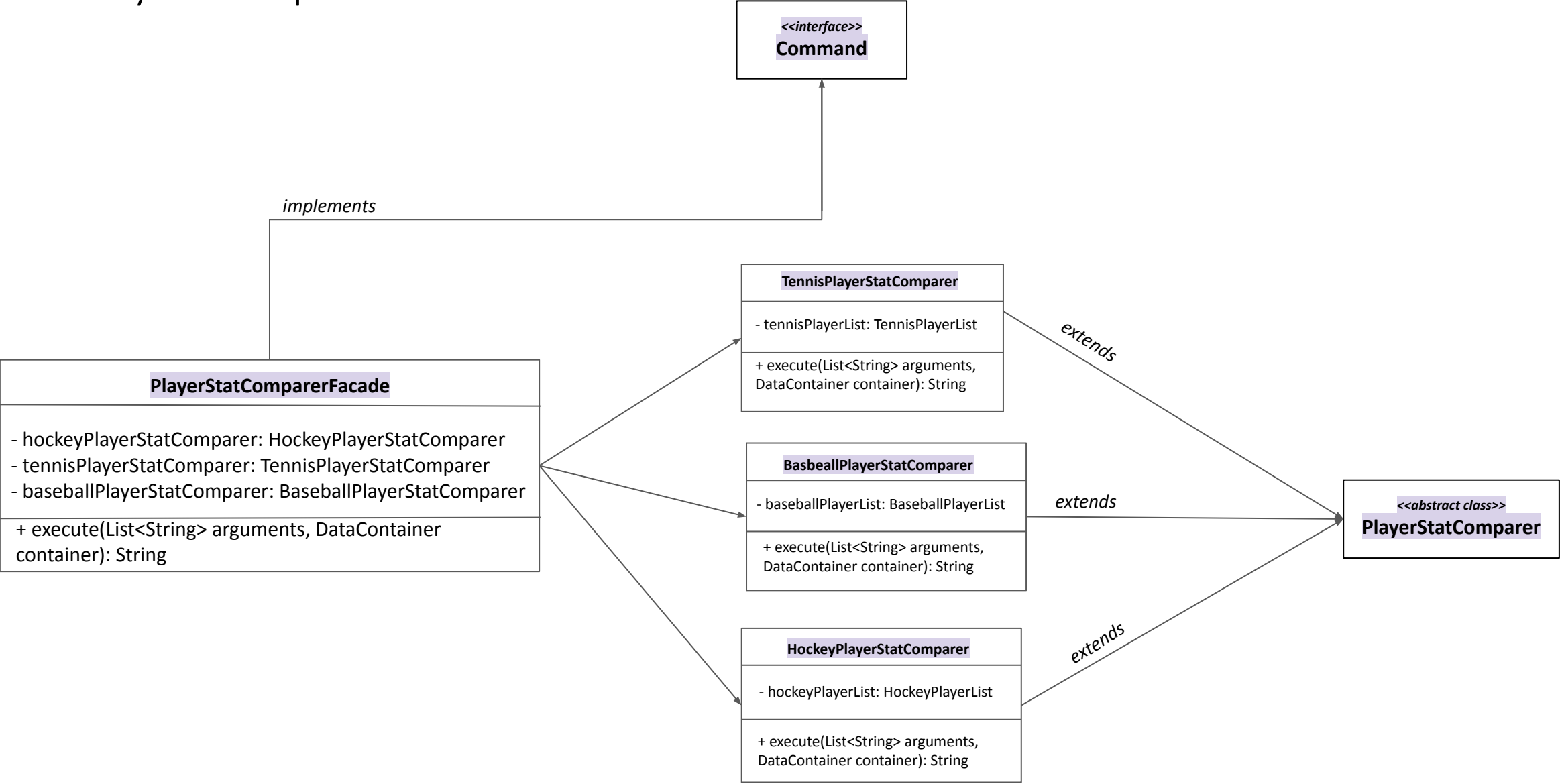
# Use Cases - Abstract Classes and Interfaces



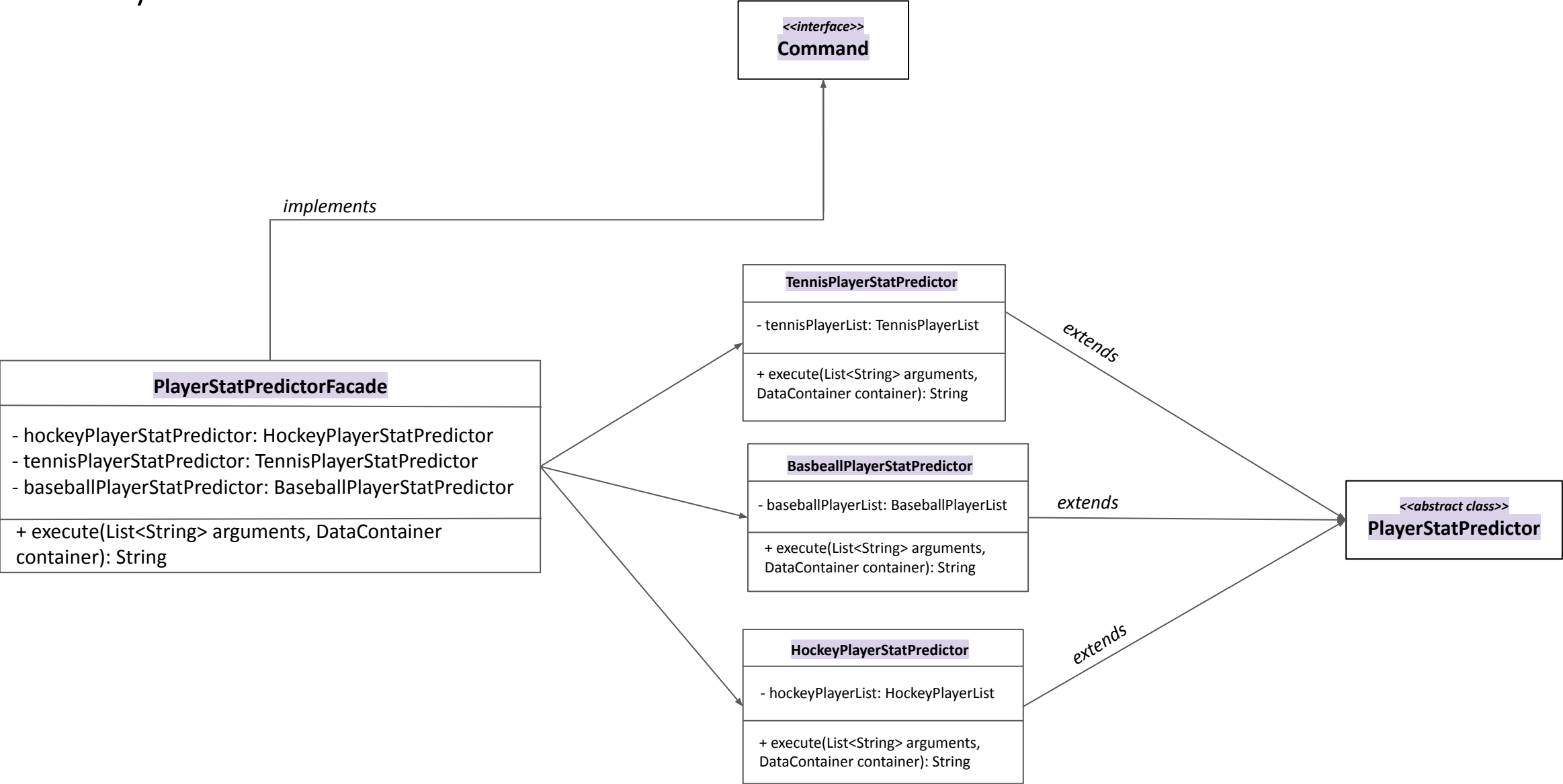
# Use Cases - PlayerStatManager



# Use Cases - PlayerStatComparer

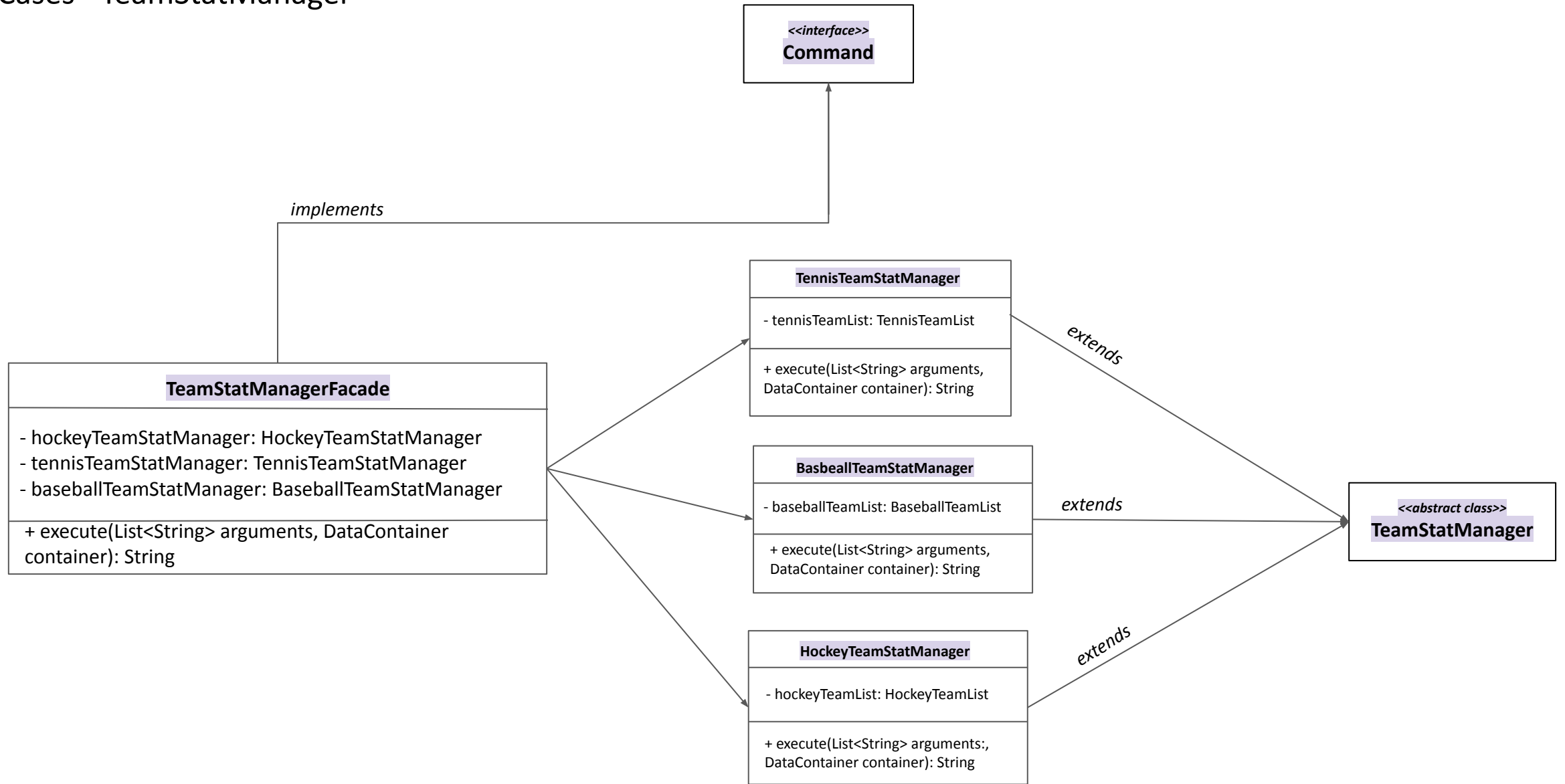


# Use Cases - PlayerStatPredictor

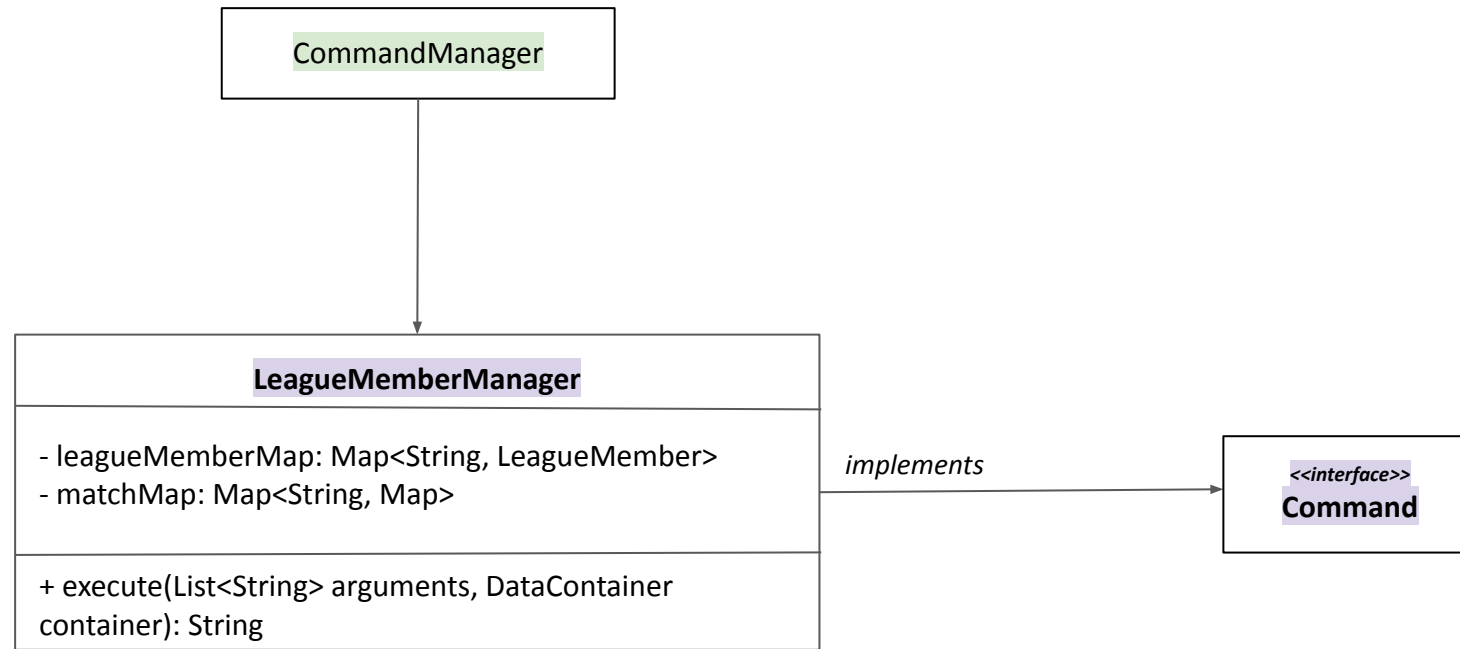




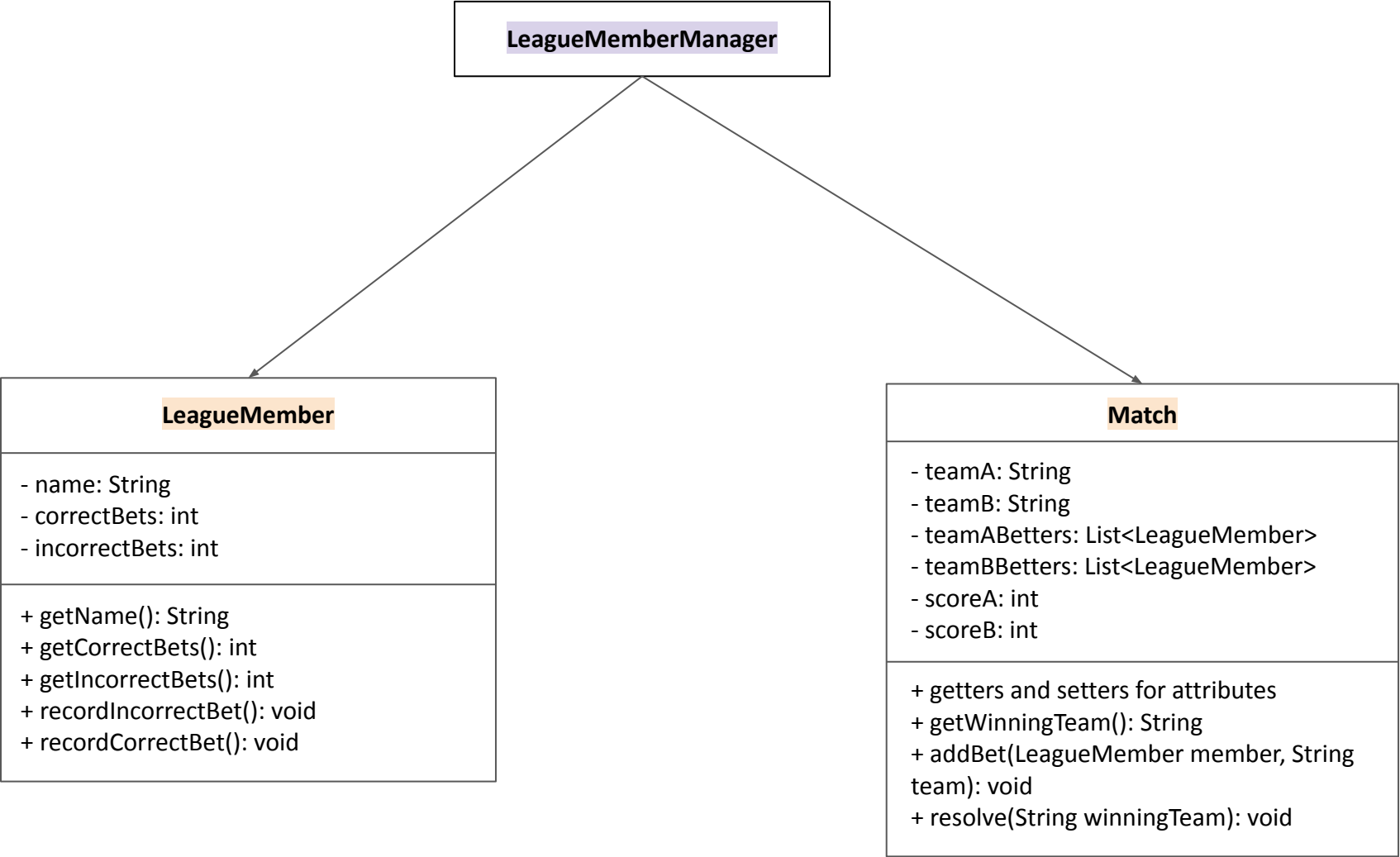
## Use Cases - TeamStatManager



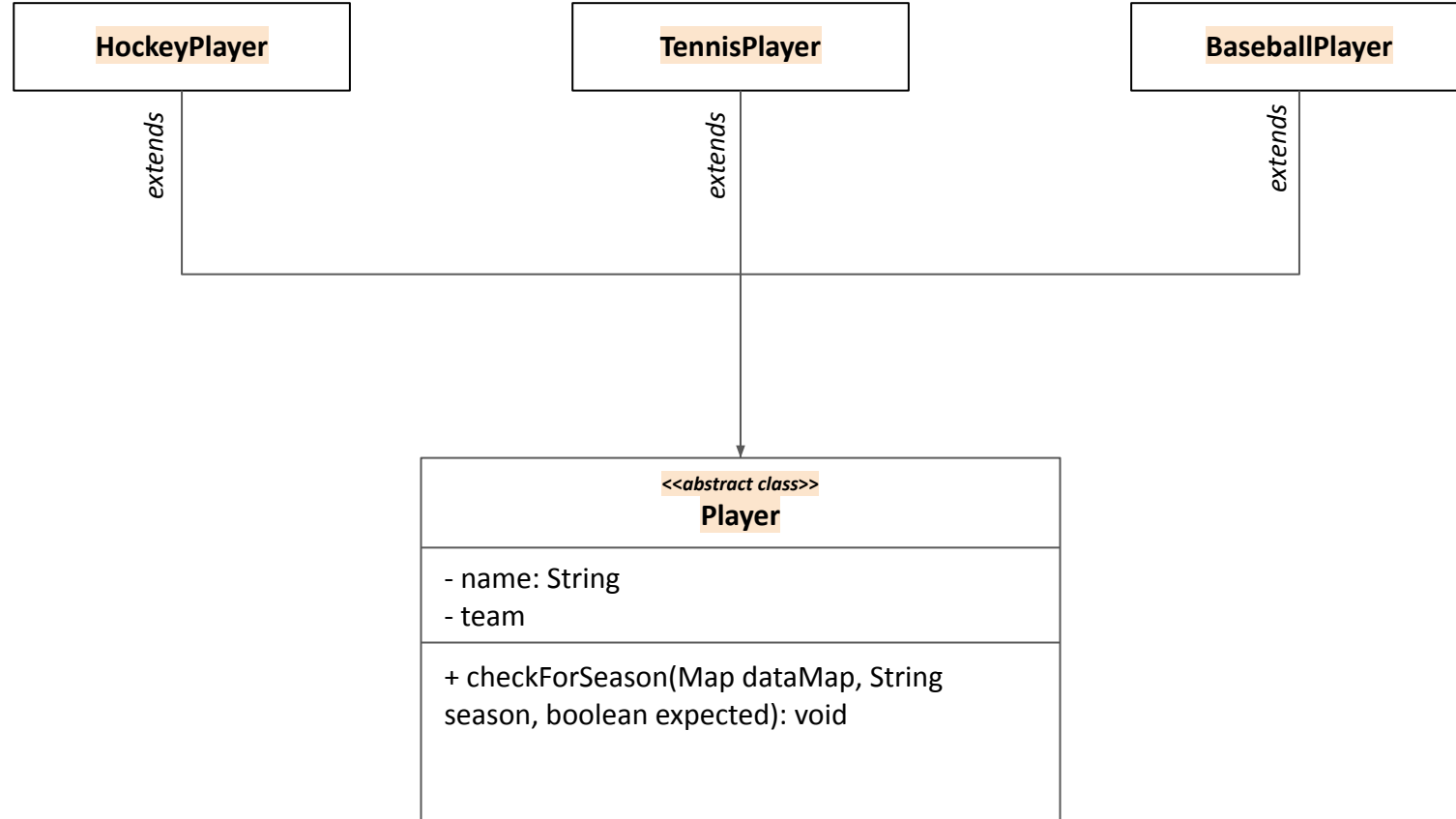
## Use Cases - LeagueMemberManager



# Entities - Fantasy League



## Entities - Player (cont.)



Entities - Team (cont.)

TeamManager
- teams: List<Team>
+ findTeamWithName(String name): Team + getTeams(): List<Team> + createTeam(Team team): void

TeamList<T extends Team>
- teamMap: Map<String, T>
+ getTeam(String name): Team + getTeams(List<String> names): List<Teams>

HockeyTeam
Relevant Hockey statistics: <ul style="list-style-type: none"><li>- Games played</li><li>- wins</li><li>- losses</li><li>- overtime losses</li><li>- faceoff win percentage</li><li>- goals for</li><li>- goals against</li><li>- shots for/against per game</li></ul>

BaseballTeam
Relevant Baseball statistics: To be Implemented

Entities - Team (cont.)

