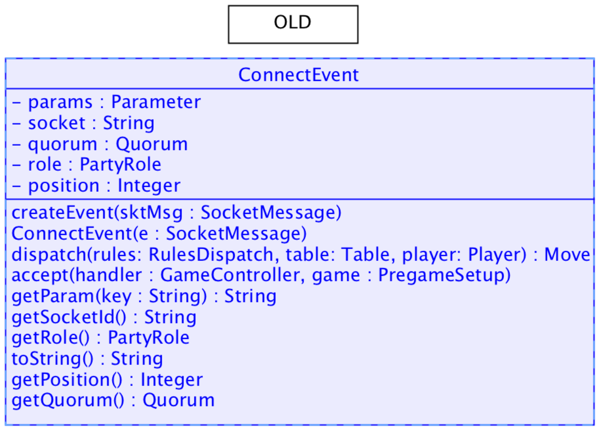
Impact Analysis

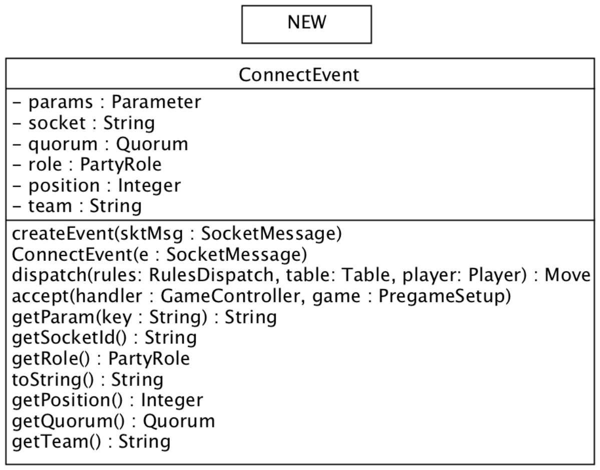
We will be adding support for team based games, in order to do this we will have to modify/create the following:

Modify ConnectEvent to capture &team="..." given in the connection string. Specifically ConnectEvent will store the team given by the player in the constructor, or set team to null if no team was given. This information will then be passed to concrete Rules classes that will dictate what is done with the team information.



```
OLD

ConnectEvent
- params : Parameter
- socket : String
- quorum : Quorum
- role : PartyRole
- position : Integer
createEvent(sktMsg : SocketMessage)
ConnectEvent(e : SocketMessage)
dispatch(rules: RulesDispatch, table: Table, player: Player) : Move
accept(handler : GameController, game : PregameSetup)
getParam(key : String) : String
getSocketId() : String
getRole() : PartyRole
toString() : String
getPosition() : Integer
getQuorum() : Quorum
```

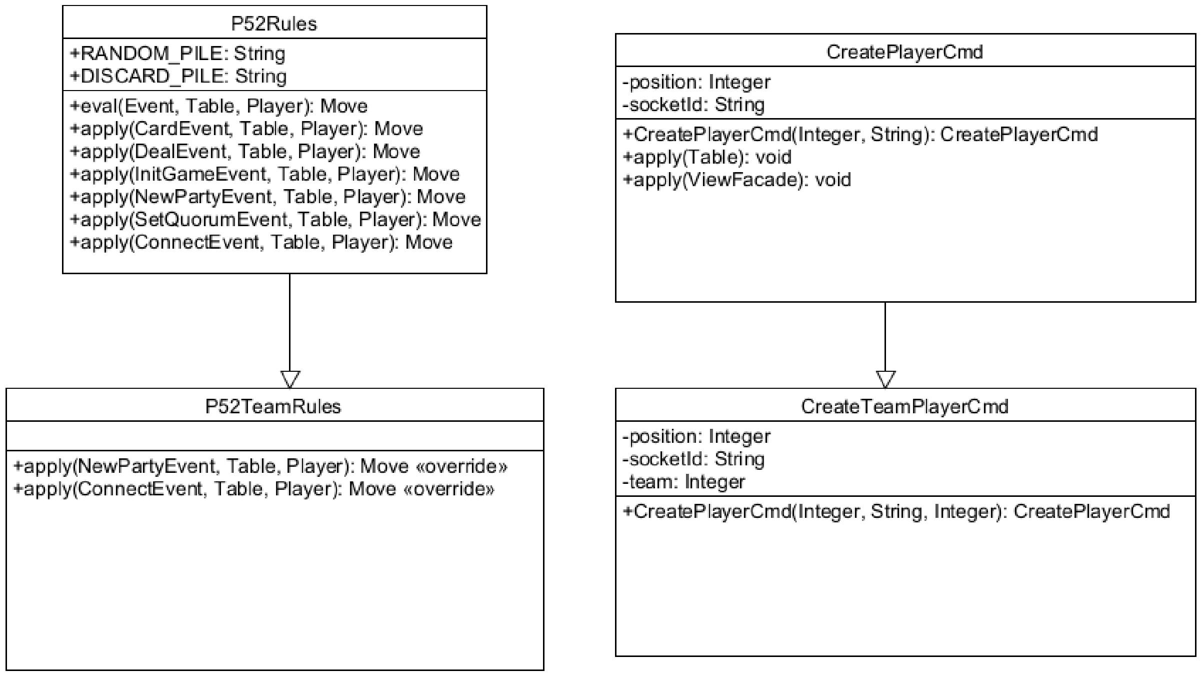


```
NEW

ConnectEvent
- params : Parameter
- socket : String
- quorum : Quorum
- role : PartyRole
- position : Integer
- team : String
createEvent(sktMsg : SocketMessage)
ConnectEvent(e : SocketMessage)
dispatch(rules: RulesDispatch, table: Table, player: Player) : Move
accept(handler : GameController, game : PregameSetup)
getParam(key : String) : String
getSocketId() : String
getRole() : PartyRole
toString() : String
getPosition() : Integer
getQuorum() : Quorum
getTeam() : String
```

Implementing a TeamTable interface with team methods that implements the Table interface would allow the methods that use Table (there are a lot of them) to continue to do so, since TeamTable is a type of table. This interface would be used for almost any team game, since it will have access to teams and their players, team scores, and the sockets of the players within a team. The functionality will be very similar to a regular table, while still technically being able to function as a regular table (since it will still have the functions that a regular Table has). Some of the methods that TeamTable will have are getTeam(Player p) and getTeams(), which can be used to get a specific team by player and get all teams. Another thing we could do is strip down the table class to have only what a table itself does (with methods like addPile(), getPile(), isMatchOver(), etc.) and create two separate kinds of tables, one being TeamTable (that includes methods like getTeams() and getTeamScore()) and one being IndividualTable (that includes methods like getPlayers() and getPlayerScore(). This would eliminate the code smell of too many methods in one class and split the functionality of a table into two separate entities that have more specific uses than just Table.

Extend the rules class to deal with the creation of teams which would be dealt with when the MatchController is setting up a game. This is so when a new player is created it will be given a team number. This would happen when the MatchController would receive a ConnectEvent as the ConnectEvent would have to specify which team the player would be on and if not the team would be chosen by the rules based on which team needs a player. After this the rules would create a new CreatePlayerCmd which would specify which team the player would be on. This would be dealt with by the TeamTable class which would be accessed through the TeamTable Interface. This would be specific to the game that was being played but would just be extended off the current rules and would only have to override things like when a NewPartyEvent would happen or when a ConnectEvent would happen to implement assigning teams when the players join. Also for this to work the CreatePlayerCmd would also have to be extended to deal with keeping track of the players team. This for something like P52rules and CreatePlayerCmd would look something like:


P52Rules
+RANDOM_PILE: String
+DISCARD_PILE: String
+eval(Event, Table, Player): Move
+apply(CardEvent, Table, Player): Move
+apply(DealEvent, Table, Player): Move
+apply(InitGameEvent, Table, Player): Move
+apply(NewPartyEvent, Table, Player): Move
+apply(SetQuorumEvent, Table, Player): Move
+apply(ConnectEvent, Table, Player): Move
P52TeamRules
+apply(NewPartyEvent, Table, Player): Move «override»
+apply(ConnectEvent, Table, Player): Move «override»
CreatePlayerCmd
-position: Integer
-socketId: String
+CreatePlayerCmd(Integer, String): CreatePlayerCmd
+apply(Table): void
+apply(ViewFacade): void
CreateTeamPlayerCmd
-position: Integer
-socketId: String
-team: Integer
+CreatePlayerCmd(Integer, String, Integer): CreatePlayerCmd

From the new Table class, we would extend it to create a TeamTableClass in addition to the IndivualTableClass. This new TeamTableClass would be similar to the previous Table implementation, but would instead have a hashmap of teams. The functions that use the player map would instead use the team map to track scores, distribute cards, and more.

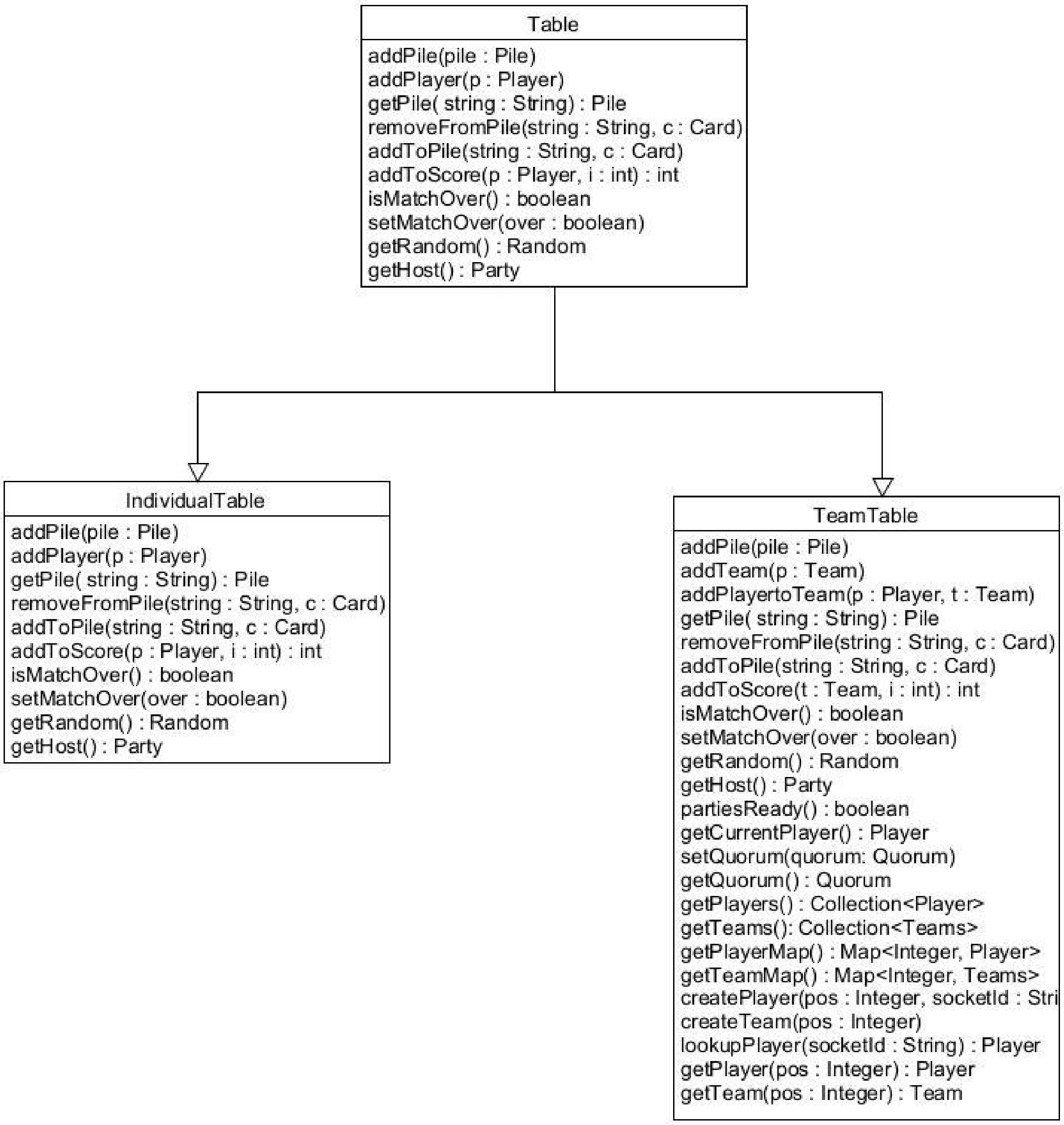
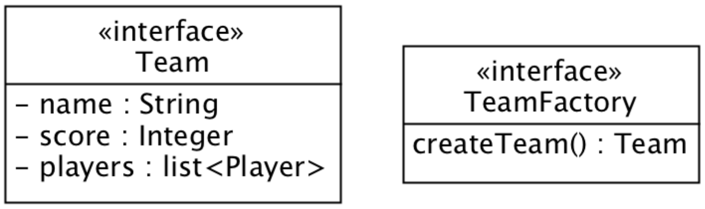


Table
addPile(pile : Pile)
addPlayer(p : Player)
getPile(string : String) : Pile
removeFromPile(string : String, c : Card)
addToPile(string : String, c : Card)
addToScore(p : Player, i : int) : int
isMatchOver() : boolean
setMatchOver(over : boolean)
getRandom() : Random
getHost() : Party
IndividualTable
addPile(pile : Pile)
addPlayer(p : Player)
getPile(string : String) : Pile
removeFromPile(string : String, c : Card)
addToPile(string : String, c : Card)
addToScore(p : Player, i : int) : int
isMatchOver() : boolean
setMatchOver(over : boolean)
getRandom() : Random
getHost() : Party
TeamTable
addPile(pile : Pile)
addTeam(p : Team)
addPlayertoTeam(p : Player, t : Team)
getPile(string : String) : Pile
removeFromPile(string : String, c : Card)
addToPile(string : String, c : Card)
addToScore(t : Team, i : int) : int
isMatchOver() : boolean
setMatchOver(over : boolean)
getRandom() : Random
getHost() : Party
partiesReady() : boolean
getCurrentPlayer() : Player
setQuorum(quorum: Quorum)
getQuorum() : Quorum
getPlayers() : Collection<Player>
getTeams(): Collection<Teams>
getPlayerMap() : Map<Integer, Player>
getTeamMap() : Map<Integer, Teams>
createPlayer(pos : Integer, socketId : Stri
createTeam(pos : Integer)
lookupPlayer(socketId : String) : Player
getPlayer(pos : Integer) : Player
getTeam(pos : Integer) : Team

add a Team interface and a TeamFactory


«interface»
Team
– name : String
– score : Integer
– players : list<Player>
«interface»
TeamFactory
createTeam() : Team