|  |
| --- |
|  |
|  | // Although there can be other ways to solve in python but this would be much easier to solve using Lodash a javascript utility library released by MIT.  **//Summary:**  As per the specs there is a Player type with properties position that is a position in current game and positions that are the positions that player took in all games of a tournament  // The Player.position has two variants: BPosition for basketball positions and HPosition for handball positions. Each position variant implements corresponding calculatePoints algorithm.  // The Team type represents team players and team totalPoints calculated in accordance with sport type. The selection of the totalPoints calculation method is done dynamically via **mixin**. There are three totalPoints calculation mixin: calculateBasketballTotalPoints, calculateHandballTotalPoints and calculateDefaultTotalPoints. Mixin is a class containing methods that can be used by other classes without a need to inherit from it.  parseBasketball and parseHandball are extensible input statistics file parsers implemented using **regular expressions** and selected dynamically depending on input file content. These parsers are the only place in the application where new objects Player, BPosition and HPosition are created. Regular expressions are patterns used to match character combinations in strings. In JavaScript, regular expressions are also objects.  // Pseudocode   1. First we will invoke the lodash utility library of the JS 2. Checking for player type 3. Checking for position variant for basketball with regards to different parameters viz. sport, game, team, position, score, rebounds, assists 4. Rules for basketball player points calculation 5. Checking for position variant for handball 6. Rules for handball player points calculation 7. ADD NEW POSITION VARIANTS WITH CORRESPONDING POINTS CACLULATION RULES 8. default mixin for Team.totalPoints calculation 9. basketball mixin for Team.totalPoints calculation 10. handball mixin for Team.totalPoints calculation 11. ADD NEW Team.totalPoints CALCULATION ALGORITHMS 12. Handball input statistics file parser 13. ADD NEW INPUT STATISTICS FILE PARSERS 14. Recalculating Player.points in corresponding position after knowing winner team 15. caclulate Team.totalPoints using corresponding mixin mixed in dynamically 16. Add 10 extra points to all players of winner team 17. calculate Most Valuable Player according to its points   // Actual code are as below:  //1. First we will invoke libraries which are as below Include the file system module  var fs = require(‘fs’);  // Load the full build in node.js  var \_ = require('lodash'); |
|  |  |
|  | //2. player type with position in each game and its all tournament positions |
|  | var Player = function(name, nickname, number) { |
|  | this.name = name || ''; |
|  | this.nickname = nickname || ''; |
|  | this.number = number || ''; |
|  | this.position = null; |
|  | this.points = 0; |
|  | this.positions = [ ]; |
|  | }; |
|  |  |
|  | //3. position variant for basketball |
|  | var BPosition = function(sport, game, team, position, score, rebounds, assists) { |
|  | this.sport = sport || ''; |
|  | this.game = game || ''; |
|  | this.team = team || ''; |
|  | this.position = position || ''; |
|  | this.score = score || 0; |
|  | this.rebounds = rebounds || 0; |
|  | this.assists = assists || 0; |
|  | this.winner = false; |
|  | }; |
|  |  |
|  | //4. basketball player points calculation rules |
|  | BPosition.prototype.calculatePoints = function() { |
|  | var rules = { |
|  | default: { score: 1, rebounds: 1, assists: 1 } |
|  | , G: { score: 2, rebounds: 3, assists: 1 } |
|  | , F: { score: 2, rebounds: 2, assists: 2 } |
|  | , C: { score: 2, rebounds: 1, assists: 3 } |
|  | }; |
|  | var rule = \_.has(rules, this.position) // \_.has(object, path) lodash will traverse using this.position as path |
|  | ? rules[this.position] : rules['default']; |
|  | return this.score \* rule.score + this.rebounds \* rule.rebounds |
|  | + this.assists \* rule.assists; |
|  | }; |
|  |  |
|  | //5. position variant for handball |
|  | var HPosition = function(sport, game, team, position, made, received) { |
|  | this.sport = sport || ''; |
|  | this.game = game || ''; |
|  | this.team = team || ''; |
|  | this.position = position || ''; |
|  | this.made = made || 0; |
|  | this.received = received || 0; |
|  | this.winner = false; |
|  | }; |
|  |  |
|  | //6. handball player points calculation rules |
|  | HPosition.prototype.calculatePoints = function() { |
|  | var rules = { |
|  | default: { initial: 10, made: 2, received: -1 } |
|  | , G: { initial: 50, made: 5, received: -2 } |
|  | , F: { initial: 20, made: 1, received: -1 } |
|  | }; |
|  | var rule = \_.has(rules, this.position) // \_.has(object, path) lodash will traverse using this.position as path |
|  | ? rules[this.position] : rules['default']; |
|  | return rule.initial + this.made \* rule.made |
|  | + this.received \* rule.received; |
|  | }; |
|  |  |
|  | // 7. ADD NEW POSITION VARIANTS WITH CORRESPONDING POINTS CACLULATION RULES |
|  |  |
|  | // team type with players and totalPoints caclulated by appropriate mixin. A  [mixin](https://en.wikipedia.org/wiki/Mixin) is a class containing methods that can be used by other classes without a need to inherit from it. |
|  | var Team = function(sport, game, name, players) { |
|  | this.sport = sport || ''; |
|  | this.game = game || ''; |
|  | this.name = name || ''; |
|  | this.players = players || [ ]; |
|  | this.totalPoints = 0; |
|  | }; |
|  |  |
|  | //8. default mixin for Team.totalPoints caclulation |
|  | var calculateDefaultTotalPoints = function() { |
|  | this.totalPoints = \_.sum(this.players, 'points'); //Gets the sum of the values in collection. |
|  | }; |
|  |  |
|  | //9. basketball mixin for Team.totalPoints caclulation |
|  | var calculateBasketballTotalPoints = function() { |
|  | this.totalPoints = \_.sum(this.players, 'points'); //Gets the sum of the values in collection. |
|  | }; |
|  |  |
|  | // 10.handball mixin for Team.totalPoints caclulation |
|  | var calculateHandballTotalPoints = function() { |
|  | this.totalPoints = \_.sum(this.players, 'position.made');// Gets the sum of the values in collection. |
|  | }; |
|  |  |
|  | // 11. ADD NEW Team.totalPoints CALCULATION ALGORITHMS |
|  |  |
|  | // Basketball input statistics file parser |
|  | var parseBasketball = function(game, rawGame) { |
|  | var players = [ ]; |
|  | var m = null; |
|  | // now use regex and saving under rplayer  var rPlayer = /^([a-z][^;]\*);([^;]+);([^;]+);([^;]+);([^;]+);(\d+);(\d+);(\d+)$/mg; |
|  | // checking for not null value  while ((m = rPlayer.exec(rawGame)) !== null) {  // saving new player position for specified index under variable player |
|  | var player = new Player(m[1], m[2], m[3]);  // saving new basket ball position for specified index under variable player |
|  | var bposition = new BPosition('basketball', game, m[4], m[5] |
|  | , parseInt(m[6]), parseInt(m[7]), parseInt(m[8])); // The parseInt() function parses a string and returns an integer. |
|  | player.position = bposition; |
|  | players.push(player);// The push() method adds new item player to the end of an array players, and returns the new length. |
|  | }; |
|  | return players; |
|  | }; |
|  |  |
|  | // 12.Handball input statistics file parser |
|  | var parseHandball = function(game, rawGame) { |
|  | var players = [ ]; |
|  | var m = null; |
|  | var rPlayer = /^([a-z][^;]\*);([^;]+);([^;]+);([^;]+);([^;]+);(\d+);(\d+)$/mg;// using regex  // checking for null values |
|  | while ((m = rPlayer.exec(rawGame)) !== null) { |
|  | // saving new player position for specified index under variable player  var player = new Player(m[1], m[2], m[3]);  // saving new hand ball position for specified index under variable player |
|  | var hposition = new HPosition('handball', game, m[4], m[5] |
|  | , parseInt(m[6]), parseInt(m[7])); |
|  | player.position = hposition; |
|  | players.push(player); // The push() method adds new item player to the end of an array players, and returns the new length. |
|  | }; |
|  | return players; |
|  | }; |
|  |  |
|  | //13. ADD NEW INPUT STATISTICS FILE PARSERS |
|  |  |
|  | // calculate Player.points in corresponding position before knowing winner team |
|  | var calculatePlayersPoints = function(players) { |
|  | var calculatePoints = function(player) { |
|  | player.points = player.position.calculatePoints(); |
|  | return player; |
|  | }; |
|  | return \_.map(players, calculatePoints);//  \_.map() Returns a new array of the results of each callback execution. |
|  | }; |
|  |  |
|  | // 14.Recalculating Player.points in corresponding position after knowing winner team |
|  | var recalculateWinnersPoints = function(players) { |
|  | var calculateTeamTotalPoints = function(players, name) { |
|  | var totalPointsMixin = { |
|  | default: calculateDefaultTotalPoints |
|  | , basketball: calculateBasketballTotalPoints |
|  | , handball: calculateHandballTotalPoints |
|  | }; |
|  | var sport = \_.first(players).position.sport;// \_.first() Returns the first element of array. |
|  | var game = \_.first(players).position.game; //\_.first() Returns the first element of array. |
|  | var team = new Team(sport, game, name, players); |
|  | var calculateTotalPoints = \_.has(totalPointsMixin, sport) |
|  | ? totalPointsMixin[sport] : totalPointsMixin['default']; |
|  | \_.mixin(team, { calculateTotalPoints: calculateTotalPoints }); //Adds all own enumerable string keyed function properties of a source object to the destination object. If object is a function, then methods are added to its prototype as well. |
|  | // 15. caclulate Team.totalPoints using corresponding mixin mixed in dynamically |
|  | team.calculateTotalPoints(); |
|  | return team; |
|  | }; |
|  | var winnerTeam = \_.chain(players).groupBy('position.team')//\_.chain()  Returns the new lodash wrapper instance, while .groupBy() Returns the composed aggregate object. |
|  | .map(calculateTeamTotalPoints).max('totalPoints').value();// .map()Returns the new mapped array, while .max() Returns the maximum value. |
|  | // 16.Add 10 extra points to all players of winner team |
|  | var addExtraPoints = function(player) { |
|  | player.points += 10; |
|  | player.position.winner = true; |
|  | }; |
|  | \_.map(winnerTeam.players, addExtraPoints); |
|  | return players; |
|  | }; |
|  |  |
|  | var processGames = function(files) { |
|  | var processGame = function(file) { |
|  | //\_.partial() Creates a function that invokes func with partials prepended to the arguments it receives. This method is like [\_.bind](https://lodash.com/docs/4.17.15#bind) except it does **not** alter the this binding and returns the partially applied function  var readGame = \_.partial(fs.readFileSync, file, 'utf8'); |
|  | var parseGame = function(rawGame) { |
|  | var parsers = { basketball: parseBasketball, handball: parseHandball }; |
|  | var m = rawGame.match(/^[A-Z]+$/m); //using regex |
|  | var sport = m ? m[0].toLowerCase() : ''; |
|  | // dynamically select appropriate parser  //.has() Returns true if path exists, else false. |
|  | return \_.has(parsers, sport) ? parsers[sport](file, rawGame) : [ ]; |
|  | };  //\_.flow()Returns the new composite function. |
|  | return \_.flow(readGame, parseGame, calculatePlayersPoints |
|  | , recalculateWinnersPoints)(); |
|  | }; |
|  | return \_.map(files, processGame); // \_.map() Returns the new mapped array. |
|  | }; |
|  | // 16. ADD NEW GAME STATS FILES HERE |
|  | //\_.partial () Returns the new partially applied function.  processGames = \_.partial(processGames |
|  | , [ 'games/basketball.txt', 'games/handball.txt' ]); |
|  |  |
|  | // aggregate players points |
|  | var aggregatePlayersPoints = function(players) { |
|  | var aggregatePoints = function(samePlayers) { |
|  | var player = \_.first(samePlayers); |
|  | // sum player points in all games |
|  | player.points = \_.sum(samePlayers, 'points'); |
|  | // save all player position during tournament |
|  | player.positions = \_.pluck(samePlayers, 'position');// **\_.pluck() function** is used when we need to extract a list of a given property |
|  | player.position = null; |
|  | return player; |
|  | }; |
|  | return \_.chain(players).flatten().groupBy('nickname').map(aggregatePoints) |
|  | .value(); //chain() Returns the new lodash wrapper instance. |
|  | }; |
|  |  |
|  | //17. calculate Most Valuable Player according to its points |
|  | var calculateMVP = \_.partial(\_.max, \_, 'points');// Returns the new partially applied function. |
|  |  |
|  | //\_.flow()Returns the new composite function, while consol.log() prints the output  \_.flow(processGames, aggregatePlayersPoints, calculateMVP, console.log)(); |