Awards

**This problem is worth 15 points.**  One point is awarded for the student tester (given with the folder), one point for correctly implementing each of the 6 methods in the Player class, two points for each of the four methods in the Award class.

Problem Description: You work for the IT department of a basketball team. Basketball has an almost infinite number of stats—what a great job for a computer programmer! You will write methods to implement six Player accessor methods and four Awards methods given a List of Players with their stats.

In this problem you are given the incomplete immutable helper class Player that is used to represent different players on a basketball team. The Player class has 4 completed accessor methods which report the Players name, number of free throws made, number of two point shots made, and number of three point shots made. You must complete six additional accessor methods.

The header for the Player constructor is:

public Player(String name, // player’s name

int ftMade, // number of free throws made

int twoShotsMade, // number of 2 point shots made

int threeShotsMade, // number of 3 point shots made

int assist, // number of assist

int dRebs, // number of defensive rebounds

int oRebs, // number of offensive rebounds

int blocks, // number of block shots

int to, // number of turnovers

int fouls, // number of fouls

int steals, // number of steals

double salary); // players salary in millions

note: definition of terms are given as needed

The first method in the Player class you must complete is the int getPointsScored(). getPointsScored returns the total points scored by this player. The total points is determined adding the number of free throws plus two times the number of two point shots made plus three times the number of three point shots made.

The following code shows the results of the getPointsScored method.

|  |  |
| --- | --- |
| The following code | Returns |
| Player bryant = new Player("Bryant", 14, 49, 10, 4, 6,  1, 3, 2, 5, 15, 41); |  |
| bryant.getPointsScored(); | 142 = 14+2\*49+3\*10 |

The second method in the Player class you must complete is the int getReboundEfficiency. getReboundEfficiency returns a measures of a player's rebounding efficiency. Rebounding efficiency is calculated by summing 3 times the number of defensive rebounds and 5 times the number of offensive rebounds.

The following code shows the results of the getReboundEfficiency method.

|  |  |
| --- | --- |
| The following code | Returns |
| Player bryant = new Player("Bryant", 14, 49, 10, 4, 6,  1, 3, 2, 5, 15, 41); |  |
| bryant.getReboundEfficiency(); | 23 = 3\*6+5\*1 |

The third method in the Player class you must complete is the int getBallControlEfficiency(). getBallControlEfficiency returns a measure of a player’s ball control efficiency. The getBallControlEfficiency is determine by summing the number of assist, blocks, and steals. Then subtract then number of turnovers and the number of fouls.

The following code shows the results of the getBallControlEfficiency method.

|  |  |
| --- | --- |
| The following code | Returns |
| Player bryant = new Player("Bryant", 14, 49, 10, 4, 6,  1, 3, 2, 5, 15, 41); |  |
| bryant.getBallControlEfficiency(); | 15 = 4+3+15-2-5 |

The fourth method in the Player class you must complete is the int getMVPrating(). getMVPrating determines and returns a players MVP rating. A players MVP rating is calculated by doubling the sum of total points scored, Rebounding efficiency, and ball control efficiency. Then subtract the max of the three values (total Points score, Rebounding efficiency, and ball control efficiency) and subtract the min of the same three values.

The following code shows the results of the getMVPrating method.

|  |  |
| --- | --- |
| The following code | Returns |
| Player bryant = new Player("Bryant", 14, 49, 10, 4, 6,  1, 3, 2, 5, 15, 41); |  |
| bryant.getMVPrating(); | 203 = 2 \* ( (14+2\*49+3\*10) +  (3\*6+5\*1) +  (4+3+15-2-5) )  - max(14+2\*49+3\*10,  3\*6+5\*1,  4+3+15-2-5)  -  min(14+2\*49+3\*10,  3\*6+5\*1,  4+3+15-2-5) |

The fifth method in the Player class you must complete is the int getAssistToTurnoverMargin(). getAssistToTurnoverMargin returns two times the number of assist minus the of number of turnovers.

The following code shows the results of the getAssistToTurnoverMargin method.

|  |  |
| --- | --- |
| The following code | Returns |
| Player bryant = new Player("Bryant", 14, 49, 10, 4, 6,  1, 3, 2, 5, 15, 41); |  |
| bryant.getAssistToTurnoverRatio(); | 6 = 2 \* 4 - 2 |

The sixth method in the Player class you must complete is the int getValueRatio(). getValueRatio returns the total points scored + getReboundEfficiency minus the product of salary and turnovers

The following code shows the results of the getValueRatio method.

|  |  |
| --- | --- |
| The following code | Returns |
| Player bryant = new Player("Bryant", 14, 49, 10, 4, 6,  1, 3, 2, 5, 15, 41); |  |
| bryant.getValueRatio(); | 83 = 14+2\*49+3\*10 + 3\*6+5\*1 – 2\*41 |

The Award class has a correctly implemented constructor which creates the instance variable myTeam, an ArrayList<Player> which is used to store the players from which the four awards are selected. In addition, four instance variables are provided to maintain the four different award winners and are initialized to null.

You will complete four methods in the Award class. You will implement the following four methods: getMVP(), getOffensivePlayer(), getDefensivePlayer(), getMostEffective(), Each of the these checks if this methods you will implement are to assign the corresponding instance variables. Four completed accessor methods are provide for returning each

The first method you are to complete is getMVP(). getMVP returns the Player who has the highest MVP rating. In case multiple players have the same MVP rating, select the Player with **minimum** salary. You may assume there will not be a tie after selecting the Player with minimum salary.

The following code shows the results of the getMVP method.

|  |  |
| --- | --- |
| The following code | Returns |
| Player bryant = new Player("Bryant", 14, 49, 10, 4, 6, 1, 3, 2, 5, 15, 41));  Player james = new Player("James", 19, 52, 9, 3, 7, 2, 2, 3, 4, 17, 43));  Player harden = new Player("Harden", 17, 40, 8, 3, 3, 4, 1, 1, 3, 16, 39));  Player durant = new Player("Durant", 13, 45, 10, 4, 7, 2, 3, 3, 4, 14, 38));  Player leonard = new Player("Leonard", 10, 42, 9, 5, 9, 1, 4, 4, 2, 13, 27));  Player curry = new Player("Curry", 21, 55, 15, 7, 8, 0, 1, 1, 4, 17, 37));  List<Player> ps = new ArrayList<Player>();  ps.add(bryant);  ps.add(james);  ps.add(harden);  ps.add(durant);  ps.add(leonard);  ps.add(curry);  lu.addPlayers(ps);  Awards aw = new Awards(); |  |
| aw.getMVP()); | curry |

\* curry MVP rating is the highest (244) making him the MVP

The second method you are to complete is getOffensivePlayer(). getOffensivePlayer must not return the Player selected by getMVP. getOffensivePlayer returns the Player who has scored the most points. In case of tie, select the Player with highest assist to turnover margin. You may assume there will not be a tie after selecting the Player with highest assist to turnover ratio.

When implementing this method, you should NOT assume that the getMVP has been invoked. That is, do not assume the MVP has been determined before invoking this method.

The following code shows the results of the getOffensivePlayer method.

|  |  |
| --- | --- |
| The following code | Returns |
| Player bryant = new Player("Bryant", 14, 49, 10, 4, 6, 1, 3, 2, 5, 15, 41));  Player james = new Player("James", 19, 52, 9, 3, 7, 2, 2, 3, 4, 17, 43));  Player harden = new Player("Harden", 17, 40, 8, 3, 3, 4, 1, 1, 3, 16, 39));  Player durant = new Player("Durant", 13, 45, 10, 4, 7, 2, 3, 3, 4, 14, 38));  Player leonard = new Player("Leonard", 10, 42, 9, 5, 9, 1, 4, 4, 2, 13, 27));  Player curry = new Player("Curry", 21, 55, 15, 7, 8, 0, 1, 1, 4, 17, 37));  List<Player> ps = new ArrayList<Player>();  ps.add(bryant);  ps.add(james);  ps.add(harden);  ps.add(durant);  ps.add(leonard);  ps.add(curry);  lu.addPlayers(ps);  Awards aw = new Awards(); |  |
| aw.getOffensivePlayer(); | james |

\* curry is the MVP and not a legal choice for Offensive Player, therefore james with most points scored (150) is the Offensive Player

The third method you are to complete is getDefensivePlayer(). getDefensivePlayer must not select the Player selected by either getMVP or getOffensivePlayer. getDefensivePlayer selects the Player who has the most steals. In case of tie, select the Player with fewest fouls. You may assume there will not be a tie after selecting the Player with the fewest fouls.

When implementing this method, you should NOT assume that the getMVP or getOffensivePlayer has been invoked. That is, do not assume the MVP or Offensive Player has been determined before invoking this method.

The following code shows the results of the getDefensivePlayer method.

|  |  |
| --- | --- |
| The following code | Returns |
| Player bryant = new Player("Bryant", 14, 49, 10, 4, 6, 1, 3, 2, 5, 15, 41));  Player james = new Player("James", 19, 52, 9, 3, 7, 2, 2, 3, 4, 17, 43));  Player harden = new Player("Harden", 17, 40, 8, 3, 3, 4, 1, 1, 3, 16, 39));  Player durant = new Player("Durant", 13, 45, 10, 4, 7, 2, 3, 3, 4, 14, 38));  Player leonard = new Player("Leonard", 10, 42, 9, 5, 9, 1, 4, 4, 2, 13, 27));  Player curry = new Player("Curry", 21, 55, 15, 7, 8, 0, 1, 1, 4, 17, 37));  List<Player> ps = new ArrayList<Player>();  ps.add(bryant);  ps.add(james);  ps.add(harden);  ps.add(durant);  ps.add(leonard);  ps.add(curry);  lu.addPlayers(ps);  Awards aw = new Awards(); |  |
| aw.getDefensivePlayerYear(); | harden |

\* curry is the MVP, james is the Offensive Player and both are not legal choices for Defensive Player,  
 therefore harden with the most steals (16) is the Defensive Player.

The fourth method you are to complete is getMostEffective().getMostEffective must not select the Player selected by either getMVP, getOffensivePlayer or getDefensivePlayer. getMostEffective selects the Player who has the highest getValueRatio. In case of tie, select the Player with the most free throws made. You may assume there will not be a tie after selecting the Player with the most free throws made.

When implementing this method, you should NOT assume that the getMVP, getOffensivePlayer or getDefensivePlayer has been invoked. That is, do not assume the MVP, Offensive Player, or Defensive Player has been determined before invoking this method.

The following code shows the results of the getMostEffective method.

|  |  |
| --- | --- |
| The following code | Returns |
| Player bryant = new Player("Bryant", 14, 49, 10, 4, 6, 1, 3, 2, 5, 15, 41));  Player james = new Player("James", 19, 52, 9, 3, 7, 2, 2, 3, 4, 17, 43));  Player harden = new Player("Harden", 17, 40, 8, 3, 3, 4, 1, 1, 3, 16, 39));  Player durant = new Player("Durant", 13, 45, 10, 4, 7, 2, 3, 3, 4, 14, 38));  Player leonard = new Player("Leonard", 10, 42, 9, 5, 9, 1, 4, 4, 2, 13, 27));  Player curry = new Player("Curry", 21, 55, 15, 7, 8, 0, 1, 1, 4, 17, 37));  List<Player> ps = new ArrayList<Player>();  ps.add(bryant);  ps.add(james);  ps.add(harden);  ps.add(durant);  ps.add(leonard);  ps.add(curry);  lu.addPlayers(ps);  Awards aw = new Awards(); |  |
| aw.getMostEffective() | bryant |

\* curry is the MVP, james is the Offensive Player and harden is the Defensive Player making all three not legal choices for Most Effective, therefore bryant with highest Value Ratio (83) is the most effective player.