Player Status

Grade settings: Maximum grade: 100
Run: Yes **Evaluate**: Yes
Automatic grade: Yes
England Carom Committee needs to know the points scored by each player for the upcoming league matches. The committee selects the players as per their points. Points are based on the number of matches won by each player. Create a Python application and help them to calculate the points.

Requirement 1: Define a 'Player' class and its required members. All the attributes should be private attributes.

Component Specification: Player (Model Class)

Type(Class)	**Attributes**	**Type of attributes**	**Values**
Player	player_name	string	User input
	no_of_matches	integer	User input
	points	float	User input
	status_category	string	Should be a calculated value

Type(Class)	**Methods**	**Responsibilities**
Player	A parameterized constructor with 3 arguments: player_name, no_of_matches, and points respectively	To initialize the private member variables like player_name, no_of_matches, points and status_category. The status_category should be initialized with its default value.
	Included necessary getters and setters for the private attributes	
	find_status_category(self)	This method is used to find the **status category** of a player and set the same to the instance variable **'status_category'.** If the points are l**ess than or equal to 0**, then set the **status_category** as **'Miserable'** If the points are between **1 and 50 (both Inclusive)**, then set

		the **status_category** as **'Tolerable'** If the points are between **51 and 75 (both Inclusive)**, then set the **status_category** as **'Satisfactory'** If the points are **above 75**, then set the **status_category** as **'Excellent'**

Requirement 2: Store the Player Details

The committee now wants to store each player's point details for their future reference. For this, you have to create a utility class called **PlayerManagement'** to perform few functionalities.

The method '**add_player_details()**' should create a '**Player'** object using a parameterized constructor. Then calculate and set the **status_category** instance variable by invoking the '**find_status_category ()**' method and add this '**Player**' object to the '**player_list'**.

Component Specification: PlayerManagement (Utility Class)

Component Name	**Type(Class)**	**Attributes**	**Type of attributes**	**Methods**	**Responsibilities**
	PlayerManagement	player_list	List		
Add Player Object	PlayerManagement			add_player_details (self,player_name, no_of_matches,points)	This method has to create a **Player** object, identify the status_category of that player and set the values to the corresponding variable. Then add this object to the **player_list**. This method need not return anything.

Requirement 3: View the player details based on the status category

The committee wants to verify the details of their players over a period of time-based on the status category they belong to.

The method '**view_player_details()**' should identify the **Player** objects from **player_list** based on the status category that passed as arguments to this method. Fetch the player's objects those belong to the status category specified, store in another list, and return the same.

Component Specification: PlayerManagement(Utility Class)

Component Name	**Type(Class)**	**Methods**	**Returns**	**Responsibilities**
View player details of a particular type	PlayerManagement	view_player_details (self,status_category)	List of Player objects that are within the status category specified.	This method has to identify the Player objects from the player_list based on the status_category entered. Fetch those player objects and add them to another list and return the same. If no players are found under this category, then return an empty list.

The file **'main.py'** is to get the inputs like the **number of players** and **each player's point details** from the user.

The player details will be in the form of a string in the following format: **player_name:no_of_mathes: points .**

Parse these player details by invoking the **add_player_details()** method in the PlayerManagement class which will perform necessary operations and then add the 'Player' object to the list.

After adding the object to the list, get the input for the **status category to search** from the user, and invoke the '**view_player_details()**' method by passing the status category entered by the user. If the list returned by this method is empty then display the message "**No**

players found in this category” else display the player details as shown in the sample input/output statements.

Note:

- In the sample Input / Output provided, the highlighted text in bold corresponds to the input given by the user and the rest of the text represent the output.
- Ensure to provide the names for classes, attributes, and methods as specified in the question description.
- Adhere to the code template, if provided and do not edit or delete the codes provided in the code template.

Sample Input 1:
Enter the no. of players: **4**
Enter player 1 details:
Shane:450:40
Enter player 2 details:
Esther:30:76
Enter player 3 details:
Jane:1500:250
Enter player 3 details:
Tom:100:70

Enter the status category to search: **Excellent**

Sample Output1:
No. of players in the specified status category:2

Player Name: Esther
No of matches: 30
Points: 76.0

Player Name: Jane
No of matches: 1500
Points: 250.0

Sample Input 2:
Enter the no. of players: **2**

Enter player 1 details:
John:50:50
Enter player 2 details:
Harry:250:400

Enter the status category to search: **Miserable**

Sample Output2:
No players found in this category

Automatic evaluation[+]

player.py

```
# Please do not change the given template.  Fill your code only in the provided places.

class Player:

    # Write the code for the parameterized constructor here
    def __init__(self,player_name,no_of_matches,points):
        self.__player_name=player_name
        self.__no_of_matches=no_of_matches
        self.__points=points
         self.__status_category=""

     def get_player_name(self):
         return self.__player_name

     def set_player_name(self,player_name):
         self.__player_name=player_name

     def get_no_of_matches(self):
         return self.__no_of_matches

     def set_no_of_matches(self,no_of_matches):
         self.__no_of_matches=no_of_matches

     def get_points(self):
         return self.__points

     def set_points(self,points):
         self.__points=points

     def get_status_category(self):
         return self.__status_category

     def set_status_category(self,status_category):
         self.__status_category=status_category



     def find_status_category(self):
         # Fill your code here
         if self.__points<=0:
             self.__status_category="Miserable"
         elif self.__points>=1 and self.__points<=60:
             self.__status_category="Tolerable"
         elif self.__points>=51 and self.__points<=75:
             self.__status_category="Satisfactory"
         else:
             self.__status_category="Excellent"


         return




```

player_management.py

```python
# Please do not alter the given template
# You can add any number of methods and attributes as you required without changing the given template

import player as pl

class PlayerManagement:
    def __init__(self):
        self.__player_list=[]


    def add_player_details(self,player_name,no_of_matches,points):
        #Fill your code here
        p_obj=pl.Player(player_name,no_of_matches,points)
        p_obj.find_status_category()
        self.__player_list.append(p_obj)
        return

    def view_player_details(self,status_category):
        #Fill your code here
        lst=[]
        for i in self.__player_list:
            if status_category==i.get_status_category():
                lst.append(i)
        return lst   #TODO CHANGE THIS RETURN VALUE

```

main.py

```python
# Please do not alter the given template

import player_management as pm

def main():
    no_players=int(input("Enter the no. of players:"))

    # Fill your code here for getting inputs, creating appropriate object and
    # invoke necessary methods as per the requirement specification.
    player_obj=pm.PlayerManagement()
    for i in range(no_players):
        print("Enter the player details ",i+1,":")
        name,no_of_matches,points=input().split(':')
        player_obj.add_player_details(name,no_of_matches,float(points))

    status_category=input("Enter the status category to search:")

    player_rec=player_obj.view_player_details(status_category)
    # Fill your code for invoking the appropriate method(s) as per the requirements
    if len(player_rec)!=0:
        print("The no. of player in this category:",len(player_rec))
        for i in player_rec:
            print("Player Name:",i.get_player_name())
            print("No of matches:",i.get_no_of_matches())
            print("Points:",i.get_points())
            # print("Status category:",i.get_status_category())

    else:
        print("No players found in this category")




if __name__=='__main__':
    main()
```

Grade

Reviewed on Tuesday, 30 November 2021, 12:17 PM by Automatic grade

Grade 100 / 100

Assessment report

```
All testcases passed.  Your code has been submitted for evaluation.
```