# C# OOP Regular Exam – 09 August 2024

A football ball in a net

Description automatically generated

**Football Manager 2024**

1. **Overview**

*In the world of football, strategic gameplay and skilled athletes take center stage. This exam focuses on the realm of football, where teams compete to win games. Through a series of tasks, you will delve into object-oriented programming in C#. From creating manager and team structures to implementing game logic, this exam provides an opportunity to showcase your understanding of OOP principles and apply them to the dynamic world of football. Get ready to unleash your coding skills and take on the challenge of building a robust Football Manager application.*

## Setup

* Upload **only the FootballManager** project in every task **except** **Unit Tests.**
* **Do not modify the interfaces or their packages.**
* Use **strong cohesion** and **loose coupling.**
* **Use inheritance and the provided interfaces wherever possible**:
  + This includes **constructors**, **method parameters,** and **return types.**
* **Do not** violate your **interface** **implementations** by adding **more public methods** in the concrete class than the interface has defined.
* Make sure you have **no public fields** anywhere.
* **Exception messages** and **output messages** can be found in the **"Utilities"** folder.
* To solve this problem use **Visual Studio 2022** and **Netcoreapp 6.0**
* **Do not use** "\r\n" **for a new line.**

## Task 1: Structure (50 points)

**For this task’s evaluation logic in the methods isn’t included.**

You are given some **interfaces**, and you have to **implement** their functionality in the **correct classes**.

There are **2** types of entities: **Manager** and **Team**. There should also be a **TeamRepository**.

### Manager

The Manager is a **base class** of any **type of manager,** and it **should not be able to be instantiated**.

#### Data

* **Name** - **string**
  + If the **Name** is **null or whitespace,** throw a new **ArgumentException** with the message:

"Manager's name cannot be null or empty."

* **Ranking – double**
  + The ranking of the manager. It **can be modified** through the **RankingUpdate()** method. Be careful with the **access modifier**, because the property should be visible for the **derived classes**.

#### Behavior

##### void RankingUpdate(double updateValue)

The **RankingUpdate()** is an abstract method that should update (**increase**/**decrease**) the **Manager's** ranking property. The method should work with negative values also (if a **negative value** is passed to the method, the **Ranking** property **should be decreased**).

* The **range** of the **Ranking** is **[0.00; 100.00], do not exceed its limits**. If any of the **range limits is exceeded**, **set the value to the closest limit**.
* Keep in mind that different types of **Manager** will implement the method differently.

#### Override ToString() method:

Overrides the existing method ToString()and modifies it, so the returned string must be in the following format:

**"{managerName}** - **{managerTypeName}** (Ranking: **{managerRanking:F2}**)**"**

#### Constructor

A **Manager** should take the following values upon initialization:

string name, double ranking

#### Child Classes

There are three concrete types of **Manager**:

##### AmateurManager

It has **an initial ranking value of 15**.

Amateur manager will **multiply the value of the parameter** passed to the **RankingUpdate() by 0.75** whether it is a positive or negative number.

The Constructor of the **AmateurManager** should take the following parameters upon initialization:

stringname

##### SeniorManager

It has **an initial ranking value of 30**.

Senior manager will **not manipulate the parameter passed to the RankingUpdate() method**.

The Constructor of the **SeniorManager** should take the following parameters upon initialization:

stringname

##### ProfessionalManager

It has **an initial ranking value of 60**.

Professional manager **multiplies the value of the parameter** passed to the **RankingUpdate() by 1.5** whether it is a positive or negative number.

The Constructor of the **ProfessionalManager** should take the following parameters upon initialization:

stringname

### Team

#### Data

* **Name** - **string**
  + If the name **is null or whitespace,** throw an **ArgumentException** with the following message:

"Team name cannot be null or empty."

* **ChampionshipPoints -** **int**
  + Set the **initial value** of the property to **zero.**
  + The points earned by the team in the championship. It **can be modified** through the   
    **GainPoints()** methods.
* **TeamManager – IManager**
  + Holds the manager of the team, set via the **SignWith()** method.
* **PresentCondition – int**
  + A **calculated** property that **returns the present condition of the team**, which is the **result of multiplying** **ChampionshipPoints** and **TeamManager.Ranking**, **floored** to the **nearest integer**
  + **If the team has no manager, return 0.**
  + **If the team has no points, return the manager's Ranking value, floored** to the **nearest integer**
  + ***Example: 10 championship points \* 18.75 manager ranking = 188 units present condition***

#### Behavior

##### void SignWith(IManager manager)

The **SignWith()** method **sets** the **TeamManager** property.

##### void GainPoints(int points)

The **GainPoints()** method **increases the ChampionshipPoints** property with **the given parameter value**.

##### void ResetPoints()

The **ResetPoints()** method **resets the** championship **points earned to 0**.

#### Override ToString() method:

Overrides the existing method ToString()and modify it, so the returned string must be in the following format:

**"**Team: **{Name}** Points: **{championshiPoints}"**

#### Constructor

A team should take the following values upon initialization:

string name

## TeamRepository

The **TeamRepository** is an **IRepository<ITeam>. Collection** for the **teams** that are created in the application.

### Properties

* **Models – IReadOnlyCollection<ITeam>**
  + Returns a readonly **collection of all teams**, created in the application.
* **Capacity – int**
  + The repository's **maximum capacity is 10 teams**.

### Behavior

**void Add(ITeam team)**

* **Adds** a new **team** to the **TeamRepository**. If a new team is about to join the championship, and the **capacity is going to be exceeded**, the operation is aborted.

**bool Remove(string name)**

* **Removes** a **team with the given name** from the **repository. Returns true** if the removal was **successful**, **otherwise** returns **false**.

**bool Exists(string name)**

* **Returns true** if a team with the given name **is already added** to the repository, **otherwise** returns **false**.

**ITeam Get(string name)**

* **Returns** a **team with the given name** from the **repository**, if there is any. Otherwise, returns **null**.

## Task 2: Business Logic (150 points)

## The Controller Class

The business logic of the program should be concentrated around several **commands**. You are given interfaces, which you have to implement in the correct classes.

**NOTE: Do not use** "\r\n" **for a new line.**

The first interface is **IController**. You must create a **Controller** class, which implements the interface and implements all of its methods. The constructor of **Controller** does not take any arguments. The given methods should have the logic described for each in the Commands section. When you create the **Controller** class, go into the **Engine** class constructor and uncomment the "this.controller = new Controller();" line.

**Data**

You need to keep track of the teams, this is why you need to create a private field in your controller class,   
for example:

* **championship – TeamRepository**

**Commands**

There are several commands, which control the business logic of the application. They are stated below.

**JoinChampionship Command**

**Parameters**

* **teamName - string**

**Functionality**

The method should **create and add** a new **team** to the **TeamRepository**.

* If the championship capacity is full (10 teams have joined the championship already), return the following message: "Championship is full!"
* If a team with the same **name** already exists, return the following message: "{**name}** has already joined the Championship."
* If the team is successfully created, add the team to the collection of teams and return: "{**name}** has successfully joined the Championship."

#### SignManager Command

##### Parameters

* **teamName - string**
* **managerTypeName** - **string**
* **managerName - string**

##### Functionality

The method should **create and set** a new **Manager** for the specified team.

* If the given **teamName** is NOT presented in the Championship, return the following message: "Team {teamName**}** does not take part in the Championship."
* If the given **managerTypeName** is not a valid Manager subclass (**AmateurManager**, **SeniorManager**, or **ProfessionalManager**), return the message: "**{managerTypeName}** is an invalid manager type for the application."
* If the team has already signed with another manager, return the message: **"**Team **{teamName}** has already signed a contract with **{currentManagerName}**.**"**
* Iterate through all teams (with managers) in the Championship to check if any team has already signed a contract with a manager with the same name. If another team has already signed a contract with the manager, return the message: **"**Manager **{managerName}** is already assigned to another team.**"**
* If none of the above cases is reached, create the correct type of **IManager**, based on the **managerTypeName**, and assign the manager to the team using the appropriate method. Return the following message: "Manager **{managerName}** is assigned to team **{teamName}**."

#### MatchBetween Command

##### Parameters

* **teamOneName - string**
* **teamTwoName - string**

##### Functionality

The **MatchBetween** method simulates a match between two teams and **determines the winner** based on their present condition:

* Check if **both teams,** identified by **teamOneName** and **teamTwoName take part** in the Championship. If **either** team **does not take part**, return the following message: **"**This match does not meet the regulation rules of the Championship.**"**
* Compare the **present condition** of the two teams. The **team in the better condition wins** the game. Increment the **ChampionshipPoints** of the **winning team by 3 points**.
  + If the winning team **has a working manager**, **increase** the winning team **manager's ranking by 5 units.**
  + If the losing team **has a working manager**, **decrease** the losing team **manager's ranking by 5 units.**
  + **Return the following message: "**Team **{winningTeamName}** wins the match against **{losingTeamName}**.**"**
* **If the present conditions of both teams** are equal, **increment** the **ChampionshipPoints** of both teams by 1 point.
  + **Return the following messase: "**The match between **{teamOneName}** and **{teamTwoName}** ends in a draw.**"**

#### PromoteTeam Command

##### Parameters

* **droppingTeamName – string**
* **promotingTeamName – string**
* **managerTypeName - string**
* **managerName - string**

##### Functionality

The **PromoteTeam** method simulates the **promotion of one team** and the **demotion of another team** in the Football Manager application. The method performs the following steps:

* Check if the team identified by **droppingTeamName** competes in the Championship (**exists** in the TeamRepository). If the team **does not exist**, return the following message: **"**Team **{droppingTeamName}** does not exist in the Championship.**"**
* If a team with the same promotingTeamName **already exists**, return the following message: "**{**name} has already joined the Championship."
* Create a **new team** with the given **promotingTeamName.**
* Iterate through all teams in the Championship to check if any team has already signed a contract with a manager with the same name. If another team has already signed a contract with the manager OR the given **managerTypeName** is not a valid Manager subclass (**AmateurManager**, **SeniorManager**, or **ProfessionalManager**), leave the new team without a manager.
* Create a **new manager**, with the given **managerName**, and **assign the new manager to the new team**.
* The promotions always happen before starting a new Championship season, so **all the teams** have their **ChampionshipPoints** reset.
* **Remove the team** with the given **droppingTeamName** from the Championship.
* Add the **new team** to compete **in the championship**, and **return the following message**: "Team {promotingTeamName} wins a promotion for the new season."

#### ChampionshipRankings Command

##### Functionality

Returns the **current ranking table of the Championship**. The teams are **ordered by** ChampionshipPoints, in **descending** order, **then by** their PresentCondition, also in **descending order**. For every team, there should be information for the team's manager. In order to receive the correct output, use the ToString() method **of each team and manager:**

"\*\*\*Ranking Table\*\*\*

1. **{team1}**/**{teamManager}**

2. **{team2}**/**{teamManager}**

3. **{team3}**/**{teamManager}**

4. **{team4}**/**{teamManager}**

**…**

'n'. **{teamn}**/**{teamManager}"**

**NOTE: Do not use** "\r\n" **for a new line.**

#### Exit Command

##### Functionality

Ends the program.

### Input / Output

You are provided with one interface, which will help you with the correct execution process of your program. The interface is Engine, and the class implementing this interface should read the input, and when the program finishes, this class should print the output.

#### Input

Below, you can see the **format** in which **each command** will be given in the input:

* **JoinChampionship** **{teamName}**
* **SignManager** **{teamName}** **{managerTypeName}** **{managerName}**
* **MatchBetween** **{teamOneName} {teamTwoName}**
* **PromoteTeam {droppingTeamName}{promotingTeamName}{managerTypeName}{managerName}**
* **ChampionshipRankings**
* **Exit**

#### Output

Print the output from each command when issued. Print the exception message if an exception is thrown during any of the commands' execution.

#### Examples

|  |
| --- |
| **Input** |
| **JoinChampionship ManchesterUnited**  **JoinChampionship ManchesterCity**  **JoinChampionship Liverpool**  **JoinChampionship Tottenham**  **JoinChampionship Arsenal**  **JoinChampionship Arsenal**  **JoinChampionship AstonVilla**  **JoinChampionship Brentford**  **JoinChampionship Brighton**  **JoinChampionship Chelsea**  **JoinChampionship Everton**  **JoinChampionship LeichesterCity**  **SignManager Liverpool ProfessionalManager JurgenKlopp**  **SignManager ManchesterUnited ProfessionalManager ErikTenHag**  **SignManager ManchesterCity ProfessionalManager JosepGuardiola**  **SignManager Tottenham LicensedManager MassimilianoAllegri**  **SignManager Arsenal SeniorManager ArsenVenger**  **SignManager AstonVilla AmateurManager ErikTenHag**  **SignManager Brentford SeniorManager ThomasFrank**  **SignManager Brighton AmateurManager FabianHurzeler**  **SignManager Chelsea ProfessionalManager EnzoMaresna**  **SignManager Everton SeniorManager SeanDyche**  **SignManager LeichesterCity ProfessionalManager SteveCooper**  **SignManager QueensParkRangers AmateurManager JamieVardy**  **SignManager Arsenal ProfessionalManager JoseMourinho**  **MatchBetween Arsenal AstonVilla**  **MatchBetween Chelsea Brighton**  **MatchBetween Everton ManchesterUnited**  **MatchBetween ManchesterCity Brentford**  **MatchBetween Liverpool Arsenal**  **MatchBetween QueensParkRangers LeichesterCity**  **MatchBetween LeichesterCity Tottenham**  **MatchBetween ManchesterUnited Everton**  **MatchBetween ManchesterCity ManchesterUnited**  **MatchBetween Brighton Brentford**  **MatchBetween AstonVilla Chelsea**  **MatchBetween Arsenal Tottenham**  **SignManager Tottenham ProfessionalManager JoseMourinho**  **SignManager AstonVilla ProfessionalManager XabiAlonso**  **MatchBetween Liverpool Chelsea**  **MatchBetween ManchesterUnited Liverpool**  **MatchBetween QueensParkRangers Liverpool**  **MatchBetween Tottenham Chelsea**  **MatchBetween Liverpool Everton**  **MatchBetween ManchesterCity Tottenham**  **MatchBetween Tottenham ManchesterCity**  **MatchBetween AstonVilla Brentford**  **MatchBetween Tottenham AstonVilla**  **MatchBetween Brighton Brentford**  **MatchBetween Everton Brighton**  **ChampionshipRankings**  **PromoteTeam LeichesterCity Redding ProfessionalManager GiovanniTrapattoni**  **PromoteTeam Everton Redding ProfessionalManager GiovanniTrapattoni**  **ChampionshipRankings**  **Exit** |
| **Output** |
| **ManchesterUnited has successfully joined the Championship.**  **ManchesterCity has successfully joined the Championship.**  **Liverpool has successfully joined the Championship.**  **Tottenham has successfully joined the Championship.**  **Arsenal has successfully joined the Championship.**  **Arsenal has already joined the Championship.**  **AstonVilla has successfully joined the Championship.**  **Brentford has successfully joined the Championship.**  **Brighton has successfully joined the Championship.**  **Chelsea has successfully joined the Championship.**  **Everton has successfully joined the Championship.**  **Championship is full!**  **Manager JurgenKlopp is assigned to team Liverpool.**  **Manager ErikTenHag is assigned to team ManchesterUnited.**  **Manager JosepGuardiola is assigned to team ManchesterCity.**  **LicensedManager is an invalid manager type for the application.**  **Manager ArsenVenger is assigned to team Arsenal.**  **Manager ErikTenHag is already assigned to another team.**  **Manager ThomasFrank is assigned to team Brentford.**  **Manager FabianHurzeler is assigned to team Brighton.**  **Manager EnzoMaresna is assigned to team Chelsea.**  **Manager SeanDyche is assigned to team Everton.**  **Team LeichesterCity does not take part in the Championship.**  **Team QueensParkRangers does not take part in the Championship.**  **Team Arsenal has already signed a contract with ArsenVenger.**  **Team Arsenal wins the match against AstonVilla.**  **Team Chelsea wins the match against Brighton.**  **Team ManchesterUnited wins the match against Everton.**  **Team ManchesterCity wins the match against Brentford.**  **Team Arsenal wins the match against Liverpool.**  **This match does not meet the regulation rules of the Championship.**  **This match does not meet the regulation rules of the Championship.**  **Team ManchesterUnited wins the match against Everton.**  **Team ManchesterUnited wins the match against ManchesterCity.**  **Team Brentford wins the match against Brighton.**  **Team Chelsea wins the match against AstonVilla.**  **Team Arsenal wins the match against Tottenham.**  **Manager JoseMourinho is assigned to team Tottenham.**  **Manager XabiAlonso is assigned to team AstonVilla.**  **Team Chelsea wins the match against Liverpool.**  **Team ManchesterUnited wins the match against Liverpool.**  **This match does not meet the regulation rules of the Championship.**  **Team Chelsea wins the match against Tottenham.**  **Team Liverpool wins the match against Everton.**  **Team ManchesterCity wins the match against Tottenham.**  **Team ManchesterCity wins the match against Tottenham.**  **Team Brentford wins the match against AstonVilla.**  **Team AstonVilla wins the match against Tottenham.**  **Team Brentford wins the match against Brighton.**  **Team Everton wins the match against Brighton.**  **\*\*\*Ranking Table\*\*\***  **1. Team: ManchesterUnited Points: 12/ErikTenHag - ProfessionalManager (Ranking: 90.00)**  **2. Team: Chelsea Points: 12/EnzoMaresna - ProfessionalManager (Ranking: 90.00)**  **3. Team: ManchesterCity Points: 9/JosepGuardiola - ProfessionalManager (Ranking: 75.00)**  **4. Team: Arsenal Points: 9/ArsenVenger - SeniorManager (Ranking: 45.00)**  **5. Team: Brentford Points: 9/ThomasFrank - SeniorManager (Ranking: 40.00)**  **6. Team: AstonVilla Points: 3/XabiAlonso - ProfessionalManager (Ranking: 60.00)**  **7. Team: Liverpool Points: 3/JurgenKlopp - ProfessionalManager (Ranking: 45.00)**  **8. Team: Everton Points: 3/SeanDyche - SeniorManager (Ranking: 20.00)**  **9. Team: Tottenham Points: 0/JoseMourinho - ProfessionalManager (Ranking: 30.00)**  **10. Team: Brighton Points: 0/FabianHurzeler - AmateurManager (Ranking: 0.00)**  **Team LeichesterCity does not exist in the Championship.**  **Team Redding wins a promotion for the new season.**  **\*\*\*Ranking Table\*\*\***  **1. Team: ManchesterUnited Points: 0/ErikTenHag - ProfessionalManager (Ranking: 90.00)**  **2. Team: Chelsea Points: 0/EnzoMaresna - ProfessionalManager (Ranking: 90.00)**  **3. Team: ManchesterCity Points: 0/JosepGuardiola - ProfessionalManager (Ranking: 75.00)**  **4. Team: AstonVilla Points: 0/XabiAlonso - ProfessionalManager (Ranking: 60.00)**  **5. Team: Redding Points: 0/GiovanniTrapattoni - ProfessionalManager (Ranking: 60.00)**  **6. Team: Liverpool Points: 0/JurgenKlopp - ProfessionalManager (Ranking: 45.00)**  **7. Team: Arsenal Points: 0/ArsenVenger - SeniorManager (Ranking: 45.00)**  **8. Team: Brentford Points: 0/ThomasFrank - SeniorManager (Ranking: 40.00)**  **9. Team: Tottenham Points: 0/JoseMourinho - ProfessionalManager (Ranking: 30.00)**  **10. Team: Brighton Points: 0/FabianHurzeler - AmateurManager (Ranking: 0.00)** |

## Task 3: Unit Tests (100 points)

You will receive a skeleton with one class inside it. **League** class will have some methods, fields, and constructors. **Cover the whole class** with the unit test to make sure that the class is working as intended.

* Your task is to **test only** the **League** **class**
* **Do NOT CHANGE OR REMOVE ANY namespaces or usings.**
* **Do not use** "\r\n" **for a new line.**
* In Judge, you upload **.zip** **(**with **Championship.Tests** inside**)** from the **skeleton.**