# C# OOP Retake Exam – 15 August 2023



**Handball**

1. **Overview**

*In the world of Handball, a thrilling team sport played by two teams of seven players, strategic gameplay and skilled athletes take center stage. This exam focuses on the exciting realm of Handball, where teams compete to score goals while defending their own nets. Through a series of tasks, you will dive into the realm of object-oriented programming in C#. From creating player and team structures to implementing game logic and player management, this exam provides an opportunity to showcase your understanding of OOP principles and apply them to the dynamic world of Handball. Get ready to unleash your coding skills and take on the challenge of building a robust Handball application.*

## Setup

* Upload **only the Handball** 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.
* For solving this problem use **Visual Studio 2019, Visual Studio 2022** and **netcoreapp 3.1, 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: **Player** and **Team**. There should also be **PlayerRepository** and **TeamRepository**.

### Player

The Player is a **base class** of any **type of player,** 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:

"Player name cannot be null or empty."

* **Rating – double**
  + The rating of the player. It **can be modified** through **IncreaseRating()** and **DecreaseRating()** methods. Be careful with the **access modifier**, because the property should be visible for the **derived classes**.
* **Team** – **string**
  + The name of the team, the player competes for. It **can be modified** only through the **JoinTeam()** method.

#### Behavior

##### void JoinTeam(string teamName)

The **JoinTeam()** method modifies the **Player’s** team property.

##### void IncreaseRating()

The **IncreaseRating()** is an abstract method that should increase the **Player’s** rating property. The **maximum value** of the **Rating** is **10, do not exceed it**. Keep in mind that different types of **Player** will implement the method differently.

##### void DecreaseRating()

The **DecreaseRating()** is an abstract method that should decrease the **Player’s** rating property. The **minimum value** of the **Rating** is **1, do not drop below**. Keep in mind that different types of **Player** 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:

**"{playerTypeName}**: **{Name}**

--Rating: **{playerRating}"**

#### Constructor

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

string name, double rating

#### Child Classes

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

##### Goalkeeper

It has **initial rating value of 2.5**.

Goalkeeper will **IncreaseRating() by 0.75** and **DecreaseRating() by 1.25**.

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

stringname

##### CenterBack

It has **initial rating value of 4**.

CenterBack will **IncreaseRating() by 1** and **DecreaseRating() by 1**.

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

stringname

##### ForwardWing

It has **initial rating value of 5.5**.

ForwardWing will **IncreaseRating() by 1.25** and **DecreaseRating() by 0.75**.

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

stringname

### Team

#### Data

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

"Team name cannot be null or empty."

* **PointsEarned -** **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 **Win()** and **Draw()** methods.
* **OverallRating – double**
  + Returns the **average rating of all players** competing in the team, **rounded to the second decimal place**. **If** the collection **does not contain any players**, **return 0**.
* **Players** - **IReadOnlyCollection<IPlayer>**
  + A readonly collection of **all players**, competing in the team.

#### Behavior

##### void SignContract(IPlayer player)

The **SignContract()** method **adds** the newcomer’s **name to the Players collection**.

##### void Win()

The **Win()** method **increases the PointsEarned** property by **3 points**. It also **increases the rating of every single player** competing for the team.

##### void Lose()

The **Lose()** method **decreases the rating of every single player** competing for the team.

##### void Draw()

The **Draw()** method **increases the PointsEarned** property by **1 point.** It also **increases the rating of the goalkeeper player** competing for the team. Each team will always have **only one goalkeeper** filed.

#### Override ToString() method:

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

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

--Overall rating: **{OverallRating}**--Players: **{name1}, {name2}…/**none**"**

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

#### Constructor

A team should take the following values upon initialization:

string name

## PlayerRepository

The **PlayerRepository** is an **IRepository<IPlayer>. Collection** for the **players** that are created in the application.

#### Data

* **Models – IReadOnlyCollection<IPlayer>**
  + Returns a readonly **collection of all players**, created in the application.

#### Behavior

**void AddModel(IPlayer player)**

* **Adds** a new **IPayer** to the PlayerRepository.

**bool RemoveModel(string name)**

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

**bool ExistsModel(string name)**

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

**IPlayer GetModel(string name)**

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

## TeamRepository

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

#### Data

* **Models – IReadOnlyCollection<ITeam>**
  + Returns a readonly **collection of all teams**, created in the application.

#### Behavior

**void AddModel(ITeam team)**

* **Adds** a new **ITeam** to the TeamRepository.

**bool RemoveModel(string name)**

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

**bool ExistsModel(string name)**

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

**ITeam GetModel(string name)**

Returns a **team with the given name** from the **collection**, 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: The Controller class SHOULD NOT handle exceptions! The tests are designed to expect messages, not exceptions!**

**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 some things, this is why you need some private fields in your controller class:

* **players – PlayerRepository**
* **teams - TeamRepository**

**Commands**

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

**NewTeam Command**

**Parameters**

* **name - string**

**Functionality**

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

* If a team with the same **name** is already added to the repository, return the following message: "{**name}** is already added to the {**correctRepositoryTypeName}**."
* If the Team is successfully created, store the team to the appropriate collection and return: "{**name}** is successfully added to the {**correctRepositoryTypeName}**."

#### NewPlayer Command

##### Parameters

* **typeName** - **string**
* **name - string**

##### Functionality

The method should **create and add** a new **IPlayer** to the **PlayerRepository**.

* If the given **typeName** is NOT presented as a valid **Player’s** child class (Goalkeeper, CenterBack or ForwardWing), return the following message: "{type**}** is invalid position for the application."
* If there is already a player with the same name, return the following message: **"{name}** is already added to the {**correctRepositoryTypeName}** as **{existingPlayerTypeName}**."
* If none of the above cases is reached, create the correct type of **IPlayer** and add it to the **PlayerRepository**. Return the following message: "{name**}** is filed for the handball league."

#### NewContract Command

##### Parameters

* **playerName - string**
* **teamName - string**

##### Functionality

The method facilitates the **signing of a contract between a player and a team** in the application. The method should perform the following steps:

* **Validates if a player with the given playerName exists in the PlayerRepository**. If no player with the provided name is found, return the following message: **"**Player with the name **{playerName}** does not exist in the {nameof(PlayerRepository)}.**"**
* **Validates if a team with the given teamName exists in the TeamRepository**. If no team with the provided name is found, return the following message: **"**Team with the name **{teamName}** does not exist in the {correctRepositoryTypeName}.**"**
* **Checks if the player has already signed a contract with a team**. If the **player.Team property is not null**, it means the player is already part of a team. In this case, return the following message: **"**Player **{playerName}** has already signed with **{player.Team}**.**"**
* **If none of the above cases is reached**, update the player.Team **property** with the **name of the team**, indicating that the player is now a part of that team. **Add the player** to the **Players collection of the team** using the appropriate method.
* **Return the following message** to confirm the successful signing of the contract: **"**Player **{playerName}** signed a contract with **{teamName}**.**"**

#### NewGame Command

##### Parameters

* **firstTeamName - string**
* **secondTeamName - string**

##### Functionality

The method simulates a game between two teams in the Handball application. It takes the **firstTeamName** and **secondTeamName** parameters as input, representing the names of the two teams. It is considered that the given **names belong to existing and already added teams**. It is considered that **both teams will have signed with enough number of players**. The method performs the following steps:

* **Retrieve the two teams**, identified by **firstTeamName** and **secondTeamName**, from the **TeamRepository. Compare the overall ratings of the two teams**.
* If the **overall rating of one of the teams is greater than the overall rating of the other team**, perform the following actions:
  + Call the **Win()** method of **the team with the greater OverallRating**.
  + Call the **Lose()** method of the **losing team**.
  + **Return** the following message: **"**Team **{winningTeamName}** wins the game over **{losingTeamName}**!**"**
* **If the overall ratings of both teams are equal**, the **game ends in a draw**. Perform the following actions:
  + Call the **Draw()** method of both teams**.**
  + **Return** the following message: **"**The game between **{firstTeamName}** and **{secondTeamName}** ends in a draw!**"**

#### PlayerStatistics Command

##### Parameters

* **teamName - string**

##### Functionality

Returns information about **each player from the team with the given name (every name passed as a parameter will be a name of an existing team in the application)**. Arrange the players by Rating - **descending**, then by Name - **alphabetically**. In order to receive the correct output, use the ToString() method **of each player:**

"\*\*\***{teamName}**\*\*\*

**{player1}**

**{player2}**

**...**

**{playern}"**

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

#### LeagueStandings Command

##### Functionality

Returns information about each team from the TeamRepository. Arrange the teams by PointsEarned - **descending**, then by OverallRating **– descending, then by teamName - alphabetically**. In order to receive the correct output, use the ToString() method **of each team:**

"\*\*\*League Standings\*\*\*

**{team1}**

**{team2}**

**...**

**{teamn}"**

**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:

* **NewTeam** **{name}**
* **NewPlayer** **{typeName}** **{name}**
* **NewContract** **{playerName} {teamName}**
* **NewGame {firstTeamName} {secondTeamName}**
* **PlayerStatistics {teamName}**
* **LeagueStandings**
* **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** |
| **NewTeam FireBall**  **NewTeam DribbleDown**  **NewTeam NetNavigators**  **NewTeam InGoodHands**  **NewTeam InGoodHands**  **NewPlayer Goalkeeper John Smith**  **NewPlayer CenterBack Al Johnson**  **NewPlayer CenterBack Bob Thompson**  **NewPlayer ForwardWing Charlie Davis**  **NewPlayer ForwardWing David Wilson**  **NewPlayer Goalkeeper Emil Brown**  **NewPlayer CenterBack Fred Clark**  **NewPlayer CenterBack Rodrigo Grade**  **NewPlayer ForwardWing Henry Lee**  **NewPlayer ForwardWing Isaac Mitchell**  **NewPlayer Goalkeeper Jack Davis**  **NewPlayer CenterBack Kyle Anderson**  **NewPlayer CenterBack Liam Taylor**  **NewPlayer ForwardWing Matthew Reed**  **NewPlayer ForwardWing Nathan Cooper**  **NewPlayer Midfielder Oliver Johnson**  **NewPlayer CenterBack John Smith**  **NewPlayer Goalkeeper Samuel Thompson**  **NewContract John Smith FireBall**  **NewContract Al Johnson FireBall**  **NewContract Bob Thompson FireBall**  **NewContract Charlie Davis FireBall**  **NewContract David Wilson FireBall**  **NewContract Emil Brown DribbleDown**  **NewContract Fred Clark DribbleDown**  **NewContract Rodrigo Grade DribbleDown**  **NewContract Henry Lee DribbleDown**  **NewContract Isaac Mitchell DribbleDown**  **NewContract Jack Davis NetNavigators**  **NewContract Kyle Anderson NetNavigators**  **NewContract Liam Taylor NetNavigators**  **NewContract Matthew Reed NetNavigators**  **NewContract Nathan Cooper NetNavigators**  **NewContract Dilan Zee InGoodHands**  **NewContract Samuel Thopmson PassFast**  **NewContract Matthew Reed NetNavigators**  **NewContract Matthew Reed DribbleDown**  **NewGame FireBall DribbleDown**  **NewGame FireBall NetNavigators**  **NewGame NetNavigators DribbleDown**  **PlayerStatistics FireBall**  **LeagueStandings**  **Exit** |
| **Output** |
| **FireBall is successfully added to the TeamRepository.**  **DribbleDown is successfully added to the TeamRepository.**  **NetNavigators is successfully added to the TeamRepository.**  **InGoodHands is successfully added to the TeamRepository.**  **InGoodHands is already added to the TeamRepository.**  **John Smith is filed for the handball league.**  **Al Johnson is filed for the handball league.**  **Bob Thompson is filed for the handball league.**  **Charlie Davis is filed for the handball league.**  **David Wilson is filed for the handball league.**  **Emil Brown is filed for the handball league.**  **Fred Clark is filed for the handball league.**  **Rodrigo Grade is filed for the handball league.**  **Henry Lee is filed for the handball league.**  **Isaac Mitchell is filed for the handball league.**  **Jack Davis is filed for the handball league.**  **Kyle Anderson is filed for the handball league.**  **Liam Taylor is filed for the handball league.**  **Matthew Reed is filed for the handball league.**  **Nathan Cooper is filed for the handball league.**  **Midfielder is invalid position for the application.**  **John Smith is already added to the PlayerRepository as Goalkeeper.**  **Samuel Thompson is filed for the handball league.**  **Player John Smith signed a contract with FireBall.**  **Player Al Johnson signed a contract with FireBall.**  **Player Bob Thompson signed a contract with FireBall.**  **Player Charlie Davis signed a contract with FireBall.**  **Player David Wilson signed a contract with FireBall.**  **Player Emil Brown signed a contract with DribbleDown.**  **Player Fred Clark signed a contract with DribbleDown.**  **Player Rodrigo Grade signed a contract with DribbleDown.**  **Player Henry Lee signed a contract with DribbleDown.**  **Player Isaac Mitchell signed a contract with DribbleDown.**  **Player Jack Davis signed a contract with NetNavigators.**  **Player Kyle Anderson signed a contract with NetNavigators.**  **Player Liam Taylor signed a contract with NetNavigators.**  **Player Matthew Reed signed a contract with NetNavigators.**  **Player Nathan Cooper signed a contract with NetNavigators.**  **Player with the name Dilan Zee does not exist in the PlayerRepository.**  **Player with the name Samuel Thopmson does not exist in the PlayerRepository.**  **Player Matthew Reed has already signed with NetNavigators.**  **Player Matthew Reed has already signed with NetNavigators.**  **The game between FireBall and DribbleDown ends in a draw!**  **Team FireBall wins the game over NetNavigators!**  **Team DribbleDown wins the game over NetNavigators!**  **\*\*\*FireBall\*\*\***  **ForwardWing: Charlie Davis**  **--Rating: 6.75**  **ForwardWing: David Wilson**  **--Rating: 6.75**  **CenterBack: Al Johnson**  **--Rating: 5**  **CenterBack: Bob Thompson**  **--Rating: 5**  **Goalkeeper: John Smith**  **--Rating: 4**  **\*\*\*League Standings\*\*\***  **Team: DribbleDown Points: 4**  **--Overall rating: 5.5**  **--Players: Emil Brown, Fred Clark, Rodrigo Grade, Henry Lee, Isaac Mitchell**  **Team: FireBall Points: 4**  **--Overall rating: 5.5**  **--Players: John Smith, Al Johnson, Bob Thompson, Charlie Davis, David Wilson**  **Team: NetNavigators Points: 0**  **--Overall rating: 2.6**  **--Players: Jack Davis, Kyle Anderson, Liam Taylor, Matthew Reed, Nathan Cooper**  **Team: InGoodHands Points: 0**  **--Overall rating: 0**  **--Players: none** |

## Task 3: Unit Tests (100 points)

You will receive a skeleton with one class inside it. **Device** 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.

* **Do NOT CHANGE OR REMOVE ANY namespaces or usings.**
* **Do not use** "\r\n" **for a new line.**
* In Judge, you upload **.zip** **(**with **SmartDevice.Tests** inside**)** from the **skeleton.**