# C# OOP Exam Preparation I



**Nautical Catch Challenge**

Exam Preparation for the ["C# OOP" course @ SoftUni".](https://softuni.bg/trainings/4377/csharp-oop-february-2024)

You can submit your solutions in [Judge](https://judge.softuni.org/Contests/4428/CSharp-OOP-Exam-Regular-09-December-2023)

1. **Overview**

*In the cozy coastal town of Meridian Bay, the waves carry whispers of the most awaited event of the year – The Nautical Catch Challenge! Eager divers, both scuba experts and free divers, gear up to plunge into the mysterious depths and shimmering shallow waters. The ocean here is home to various unique fish, each more enticing than the other. Each fish has its value, and the divers aim to fill their bags with the most prized ones. As the sun shines on the glittering sea, the competition heats up. Who will emerge as the champion diver of Meridian Bay? It’s a blend of strategy, skill, and a touch of oceanic luck.*

## Setup

* Upload **only the NauticalCatchChallenge** 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 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: **Diver** and **Fish**. There should also be a **DiverRepository** and **FishRepository**.

### Fish

The Fish is a **base class** of any **type of fish,** 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 following message:

"Fish name should be determined."

* **Points -** **double**
  + Represents the **points a fish will bring to the diver**.
  + Must be a value **between 1 and 10**, both inclusive. If not, throw a new **ArgumentException** with the message: "Points must be a numeric value, between 1 and 10."
  + This number will have **at most one decimal place**. This means the value can range from a whole number like 1 or 10, to a number with one digit after the decimal point, such as 1.1, 2.5, 3.7, or 9.1.
* **TimeToCatch – int**
  + A numerical value representing **how many seconds a diver requires to catch the fish**.

#### Override ToString() method:

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

**"{typeName}**: **{Name}** [ Points: **{Points}**, Time to Catch: **{TimeToCatch}** seconds ]**"**

#### Constructor

A Fish should take the following values upon initialization:

string name, double points, int timeToCatch

#### Child Classes

There are three concrete types of Fish:

**ReefFish**

It has **TimeToCatch value of 30 seconds**.

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

stringname, doublepoints

**PredatoryFish**

It has **TimeToCatch value of 60 seconds**.

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

string name, doublepoints

**DeepSeaFish**

It has **TimeToCatch value of 180 seconds**.

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

string name, doublepoints

### Diver

The Diver is a **base class** of any **type of diver,** 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:

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

* **OxygenLevel – int**
  + A numerical value representing the diver's **oxygen level remaining, in seconds**. If it drops **below zero** during any operation, it should be **reset to zero**.
* **Catch – IReadOnlyCollection<string>**
  + It will store a sequence of **names of fish**, caught by a specific diver.
* **CompetitionPoints – double**
  + Set the **initial value** of the property to **zero**. Returns a floating-point number **rounded** **to the** **first decimal place**. *Represents the* ***total points accumulated*** *by a diver, based on the type of fish caught during the competition*.
* **HasHealthIssues – bool**
  + The property indicates if the diver has potential health concerns. Its **initial value is False**, representing that the diver starts in a healthy state.

#### Behavior

##### void Hit(IFish fish)

The Hit method signifies a diver's attempt to catch a specific fish. When invoked with an **IFish** object as its parameter, the method performs the following actions:

* The diver's **OxygenLevel** is decreased by the value of the **TimeToCatch** property of the provided **Fish**.
* The **Name** of the caught **Fish** is added to the diver's **Catch** list.
* The diver's **CompetitionPoints** are incremented by the value of the **Points** property of the caught **Fish**.

##### void Miss(int timeToCatch)

The **Miss()** is an abstract method that should decrease the diver's **OxygenLevel** property. When the method is invoked the diver's **OxygenLevel** is decreased by **a certain value**, that will **depend on the fish that is chased**.

OxygenLevel -= (int)Math.Round(…, MidpointRounding.AwayFromZero);

##### void RenewOxy

It should be **abstract method**. The diver's **OxygenLevel** should be fully **replenished to its original or maximum value**. This would mean setting the **OxygenLevel** back to its starting value **depending on the diver’s type**.

##### void UpdateHealthStatus()

This method **changes the health status** of the diver to **True, if it is False or reciprocally**.

#### Override ToString() method:

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

**"**Diver [ Name: **{Name}**, Oxygen left: **{OxygenLevel}**, Fish caught: **{count}**, Points earned: **{CompetitionPoints}** ]**"**

#### Constructor

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

string name, int oxygenLevel

#### Child Classes

There are two concrete types of **Diver**:

##### FreeDiver

It has **OxygenLevel value of 120 seconds**.

FreeDiver will **decrease the OxygenLevel property by 60%** (using the Miss() method) of the **TimeToCatch** value **of the missed fish.**

* **If the calculated value is not a whole number, round it to the nearest whole integer.**

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

stringname

##### ScubaDiver

It has **OxygenLevel value of 540 seconds**.

ScubaDiver will **decrease the OxygenLevel property by 30%** (using the Miss() method) of the **TimeToCatch** value **of the missed a fish.**

* **If the calculated value is not a whole number, round it to the nearest whole integer.**

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

stringname

## DiverRepository

The **DiverRepository** is an **IRepository<IDiver>. Collection** for the **divers**.

### Data

* **Models – IReadOnlyCollection<IDiver>**
  + Returns a readonly **collection of all divers**, added to the repository.

### Behavior

**void AddModel(IDiver diver)**

* **Adds** a new **IDiver** to the DiverRepository.

**IDiver GetModel(string name)**

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

## FishRepository

The **FishRepository** is an **IRepository<IFish>. Collection** for the **fish**.

### Data

* **Models – IReadOnlyCollection<IFish>**
  + Returns a readonly **collection of all fish**, added to the repository.

### Behavior

**void AddModel(IFish fish)**

* **Adds** a new **IFish** to the FishRepository.

**IFish GetModel(string name)**

Returns a **fish 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: 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:

**Example:**

* **divers - DiverRepository**
* **fish - FishRepository**

**Commands**

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

**DiveIntoCompetition Command**

**Parameters**

* **diverType – string**
* **diverName - string**

**Functionality**

The method should **create and add** a new **IDiver** to the **DiverRepository**.

* If the given **diverType** is NOT presented as a valid **Diver’s** child class (FreeDiver, ScubaDiver), return the following message: "{diverType**}** is not allowed in our competition."
* If a diver with the same **Name** is already added to the repository, do not duplicate records, return the following message: "{**diverName}** is already a participant -> {**correctRepositoryTypeName}**."
* If none of the above cases is reached, the IDiver is successfully created. Store the diver to the appropriate collection and return: "{**diverName}** is successfully registered for the competition -> {**correctRepositoryTypeName}**."

#### SwimIntoCompetition Command

##### Parameters

* **fishType - string**
* **fishName - string**
* **points - double**

##### Functionality

The method should **create and add** a new **IFish** to the **FishRepository**. The **SwimIntoCompetition** method is responsible for **allowing a new fish to chase** into the competition.

* If the given **typeName** is NOT presented as a valid **Fish's** child class (ReefFish, DeepSeaFish, or PredatoryFish), return the following message: "{fishType**}** is forbidden for chasing in our competition."
* If a fish with the same **Name** is already added to the repository, do not duplicate records, return the following message: "{**fishName}** is already allowed -> {**correctRepositoryTypeName}**."
* If the above case is not reached, create the correct type of **IFish** and add it to the appropriate collection. Return the following message: "{fishName**}** is allowed for chasing."

#### ChaseFish Command

##### Parameters

* **diverName - string**
* **fishName – string**
* **isLucky - bool**

##### Functionality

The method is responsible for allowing the **specific diver** to chase and attempt **to** **catch a specific fish**:

* **Validates if a diver with the given diverName exists in the DiverRepository**. If no diver with the provided name is found, return the following message: **"**{correctRepositoryTypeName} has no **{diverName}** registered for the competition.**"**
* **Validates if a fish with the given fishName exists in the FishRepository**. If no fish with the provided name is found, return the following message: **"{fishName}** is not allowed to be caught in this competition.**"**
* **HealthCheck** - If the diver's **HasHealthIssues** property is **True**, the diver is **not allowed to dive**. Return the following message: **"{diverName}** will not be allowed to dive, due to health issues.**"**
* If the diver’s **OxygenLevel** is **less than** the fish's **TimeToCatch** value, the fish will escape, the **diver will Miss** with the harpoon (*method Miss(int timeToCatch) should be used*), and the following message should be returned: **"{diverName}** missed a good **{fishName}**.**"**
* If the diver's **OxygenLevel** is equal to the fish's **TimeToCatch** value then:
  + If **isLucky** is **True**, the diver successfully catches the fish by **invoking the Hit method** with the targeted fish. The following message is returned: **"{diverName}** hits a **{fishPoints}**pt. **{fishName}**.**"**
  + If **isLucky** is **False**, the diver misses the fish. **Invoke the Miss method** with the time to catch of the targeted fish. The following message is returned: **"{diverName}** missed a good **{fishName}**.**"**
* If the diver’s **OxygenLevel** is **more than** the fish's **TimeToCatch** value, the fish is caught, the **diver will Hit** with the harpoon, and the following message should be returned: **"{diverName}** hits a **{fishPoints}**pt. **{fishName}**.**"**
* **Please note** that if, at any point during the chase, the diver's **OxygenLevel** **drops to 0**, the diver's **HasHealthIssues** property is set to **True**.

#### HealthRecovery Command

##### Functionality

The method doesn't receive any parameters. Its main purpose is to scan through the collection of divers and identify those facing health issues:

Once the method identifies a diver with the **HasHealthIssues** property set to **True**, it initiates a series of actions to stabilize the diver:

* First, it sets the **HasHealthIssues** property of the diver to **False**, indicating that the diver is now in a stable condition.
* Secondly, it replenishes the diver's **OxygenLevel** **back to its maximum**, ensuring the divers are ready for the next dive when they feel comfortable.
* **Returns** the following message: **"**Divers recovered: **{count}"**

#### DiverCatchReport Command

##### Parameters

* **diverName - string**

##### Functionality

Returns detailed information about **a specific diver and his catch so far.** To receive the correct output, use the ToString() method **of each fish:**

"Diver [ Name: **{Name}**, Oxygen left: **{OxygenLevel}**, Fish caught: **{count}**, Points earned: **{CompetitionPoints}** ]

Catch Report:

**{fish1} //{typeName}**: **{Name}** [ Points: **{Points}**, Time to Catch: **{TimeToCatch}** seconds ]

**{fish2}**

**…**

**{fishn}**

**"**

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

#### CompetitionStatistics Command

##### Functionality

Returns information about each diver from the DiverRepository. Arrange the divers by CompetitionPoints - **descending**, then by Catch.Count **– descending, then by** Name **- alphabetically**. Return **only divers that are in good health condition**. To receive the correct output, use the ToString() method **of each diver:**

"\*\*Nautical-Catch-Challenge\*\*

**{diver1}**

**{diver2}**

**...**

**{divern}"**

**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:

* **DiveIntoCompetition** **{diverType} {diverName}**
* **SwimIntoCompetition** **{fishType}** **{fishName} {points}**
* **ChaseFish** **{diverName} {fishName} {isLucky}**
* **HealthRecovery**
* **DiverCatchReport {diverName}**
* **CompetitionStatistics**
* **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** |
| **DiveIntoCompetition ScubaDiver MaxineHarper**  **DiveIntoCompetition FreeDiver JamalCarter**  **DiveIntoCompetition SkyDiver FionaBennett**  **DiveIntoCompetition FreeDiver OscarWallace**  **DiveIntoCompetition ScubaDiver LilaMoreno**  **DiveIntoCompetition FreeDiver LilaMoreno**  **SwimIntoCompetition ReefFish AzureDamselfish 8.7**  **SwimIntoCompetition ReefFish BluestripeSnapper 6.3**  **SwimIntoCompetition ReefFish YellowtailSurgeonfish 5.0**  **SwimIntoCompetition PredatoryFish Barracuda 9.2**  **SwimIntoCompetition PredatoryFish Coryphaena 9.7**  **SwimIntoCompetition PredatoryFish Bluefish 4.4**  **SwimIntoCompetition DeepSeaFish SwordFish 10.0**  **SwimIntoCompetition DeepSeaFish Mahi-Mahi 9.1**  **SwimIntoCompetition DeepSeaFish Tuna 8.5**  **SwimIntoCompetition AquariumFish SilverArowana 3.3**  **SwimIntoCompetition DeepSeaFish Barracuda 8.6**  **ChaseFish FionaBennett AzureDamselfish False**  **ChaseFish JamalCarter SilverArowana True**  **ChaseFish MaxineHarper YellowtailSurgeonfish False**  **ChaseFish MaxineHarper AzureDamselfish True**  **ChaseFish MaxineHarper Mahi-Mahi False**  **ChaseFish MaxineHarper Tuna False**  **ChaseFish MaxineHarper Bluefish True**  **ChaseFish MaxineHarper Coryphaena True**  **ChaseFish MaxineHarper BluestripeSnapper True**  **ChaseFish OscarWallace Barracuda False**  **ChaseFish OscarWallace YellowtailSurgeonfish False**  **ChaseFish OscarWallace Tuna True**  **ChaseFish OscarWallace AzureDamselfish True**  **ChaseFish JamalCarter Tuna True**  **ChaseFish JamalCarter BluestripeSnapper False**  **ChaseFish LilaMoreno Tuna False**  **ChaseFish LilaMoreno Mahi-Mahi False**  **ChaseFish LilaMoreno SwordFish True**  **HealthRecovery**  **ChaseFish LilaMoreno Tuna False**  **ChaseFish LilaMoreno Mahi-Mahi False**  **ChaseFish LilaMoreno SwordFish True**  **DiverCatchReport LilaMoreno**  **CompetitionStatistics**  **Exit** |
| **Output** |
| **MaxineHarper is successfully registered for the competition -> DiverRepository.**  **JamalCarter is successfully registered for the competition -> DiverRepository.**  **SkyDiver is not allowed in our competition.**  **OscarWallace is successfully registered for the competition -> DiverRepository.**  **LilaMoreno is successfully registered for the competition -> DiverRepository.**  **LilaMoreno is already a participant -> DiverRepository.**  **AzureDamselfish is allowed for chasing.**  **BluestripeSnapper is allowed for chasing.**  **YellowtailSurgeonfish is allowed for chasing.**  **Barracuda is allowed for chasing.**  **Coryphaena is allowed for chasing.**  **Bluefish is allowed for chasing.**  **SwordFish is allowed for chasing.**  **Mahi-Mahi is allowed for chasing.**  **Tuna is allowed for chasing.**  **AquariumFish is forbidden for chasing in our competition.**  **Barracuda is already allowed -> FishRepository.**  **DiverRepository has no FionaBennett registered for the competition.**  **SilverArowana is not allowed to be caught in this competition.**  **MaxineHarper hits a 5pt. YellowtailSurgeonfish.**  **MaxineHarper hits a 8.7pt. AzureDamselfish.**  **MaxineHarper hits a 9.1pt. Mahi-Mahi.**  **MaxineHarper hits a 8.5pt. Tuna.**  **MaxineHarper hits a 4.4pt. Bluefish.**  **MaxineHarper hits a 9.7pt. Coryphaena.**  **MaxineHarper will not be allowed to dive, due to health issues.**  **OscarWallace hits a 9.2pt. Barracuda.**  **OscarWallace hits a 5pt. YellowtailSurgeonfish.**  **OscarWallace missed a good Tuna.**  **OscarWallace will not be allowed to dive, due to health issues.**  **JamalCarter missed a good Tuna.**  **JamalCarter missed a good BluestripeSnapper.**  **LilaMoreno hits a 8.5pt. Tuna.**  **LilaMoreno hits a 9.1pt. Mahi-Mahi.**  **LilaMoreno hits a 10pt. SwordFish.**  **Divers recovered: 4**  **LilaMoreno hits a 8.5pt. Tuna.**  **LilaMoreno hits a 9.1pt. Mahi-Mahi.**  **LilaMoreno hits a 10pt. SwordFish.**  **Diver [ Name: LilaMoreno, Oxygen left: 0, Fish caught: 6, Points earned: 55.2 ]**  **Catch Report:**  **DeepSeaFish: Tuna [ Points: 8.5, Time to Catch: 180 seconds ]**  **DeepSeaFish: Mahi-Mahi [ Points: 9.1, Time to Catch: 180 seconds ]**  **DeepSeaFish: SwordFish [ Points: 10, Time to Catch: 180 seconds ]**  **DeepSeaFish: Tuna [ Points: 8.5, Time to Catch: 180 seconds ]**  **DeepSeaFish: Mahi-Mahi [ Points: 9.1, Time to Catch: 180 seconds ]**  **DeepSeaFish: SwordFish [ Points: 10, Time to Catch: 180 seconds ]**  **\*\*Nautical-Catch-Challenge\*\***  **Diver [ Name: MaxineHarper, Oxygen left: 540, Fish caught: 6, Points earned: 45.4 ]**  **Diver [ Name: OscarWallace, Oxygen left: 120, Fish caught: 2, Points earned: 14.2 ]**  **Diver [ Name: JamalCarter, Oxygen left: 120, Fish caught: 0, Points earned: 0 ]** |

## Task 3: Unit Tests (100 points)

You will receive a skeleton with one class inside it. **RailwayStation** 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 **Railway.Tests** inside**)** from the **skeleton.**