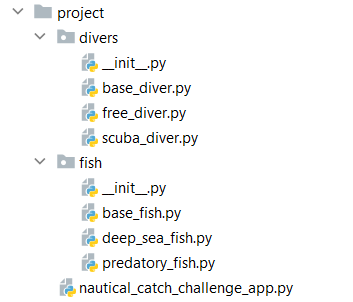
# Python OOP: Exam Preparation

**Please submit your solutions (source code) in** [**Judge**](https://judge.softuni.org/Contests/4470/Python-OOP-Regular-Exam-9-December)**.**



*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.*

***Note: You are not allowed to change the folder and file structure and their names!***



**Judge Upload**

For the **first two problems**, create a **zip** file with the **project** **folder** and **upload it** to the judge system.

For the **last problem**, create a **zip** file with the **test folder** and **upload it** to the judge system.

You do not need to include **your venv, .idea, pycache, and \_\_MACOSX (for Mac users) in the zip file**, so you do not exceed **the maximum allowed size** of **16.00 KB**.

# Structure (Problem 1) and Functionality (Problem 2)

Our task is to implement all the classes' structure and functionality (properties, methods, inheritance, abstraction, etc.)

You are **free to add additional attributes** (instance attributes, class attributes, methods, dunder methods, etc.) to simplify your code and increase readability as long as it does not change the project's final result in accordance with its requirements so that the program works properly.

## Class BaseFish

In the **base\_fish.py** file, the class **BaseFish** should be implemented. It is a **base class** for any **type of fish**,and it **should not be able to be instantiated**.

### Structure

The class should have the following attributes:

* **name:** str
  + The value represents the **name of the fish**.
  + If the name is **an empty string or contains only white spaces**, raise a ValueError with the message: **"Fish name should be determined!"**
* **points:** float
  + Represents the **points a fish will bring to the diver,** based on the **type of fish**.
  + Must be a value **between 1 and 10**, both inclusive. If not, raise a ValueErrorwith the message: **"Points should be a value ranging from 1 to 10!"**
* **time\_to\_catch:** int
  + The value represents **how many seconds a diver requires to catch the fish**.

### Methods

#### \_\_init\_\_( name: str, points: float, time\_to\_catch: int)

* In the **\_\_init\_\_** method, all the needed attributes must be set.

**fish\_details()**

* **Returns** a string with **information** about the fish depending on its type.
* Keep in mind that **each type of fish** implements the method.

## Class PredatoryFish

In the **predatory\_fish.py** file, the class **PredatoryFish** should be implemented. A predatory fish is a **type of fish**. Each predatory fish has **a time to catch of 90 seconds**.

### Methods

#### \_\_init\_\_(name: str, points: float)

* In the **\_\_init\_\_** method, all the needed attributes must be set.

**fish\_details()**

* Provide information about the **PredatoryFish** in the following format:

**"{type of fish}: {name} [Points: {points}, Time to Catch: {time\_to\_catch} seconds]"**

## Class DeepSeaFish

In the **deep\_sea\_fish.py** file, the class **DeepSeaFish** should be implemented. A deep-sea fish is a **type of fish**. Each deep-sea fish has **a time to catch of 180 seconds**.

### Methods

#### \_\_init\_\_(name: str, points: float)

* In the **\_\_init\_\_** method, all the needed attributes must be set.

**fish\_details()**

* Provide information about the **DeepSeaFish** in the following format:

**"{type of fish}: {name} [Points: {points}, Time to Catch: {time\_to\_catch} seconds]"**

## Class BaseDiver

In the **base\_diver.py** file, the class **BaseDiver** should be implemented. It is a **base class** for any **type of divers**, and it **should not be able to be instantiated**.

### Structure

The class should have the following attributes:

* **name:** str
  + The value represents the **name of the diver**.
  + If the name is **an empty string or contains only white spaces**, raise a ValueError with the message: **"Diver name cannot be null or empty!"**
* **oxygen\_level:** float
  + Represents the diver's **oxygen level remaining, in seconds**.
  + If the oxygen level **is less than 0**, raise a ValueErrorwith the message:

**"Cannot create diver with negative oxygen level!"**

* **catch: list**
  + It will store a sequence of **fish**, caught by a specific diver.
* **competition\_points:** float
  + Represents the **total points accumulated** by a diver, based on the type of fish caught during the competition. Set the **initial value** of the property to **zero**. Returns a **floating-point number** rounded to the **first decimal place**.
* **has\_health\_issue: 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.

### Methods

#### \_\_init\_\_(name: str, oxygen\_level: float)

* In the **\_\_init\_\_** method, all the needed attributes must be set.

#### miss(time\_to\_catch: int)

* Decreases the diver's **oxygen\_level** property. When the method is invoked the diver's **oxygen\_level** is decreased by **a certain value**, that will **depend on the fish that is chased**.
* Keep in mind that **each type of diver** can implement the **method differently**.

#### renew\_oxy()

* The diver's **oxygen\_level** should be fully **replenished to its original value**. This would mean setting the **oxygen\_level** back to its starting value **depending on the diver’s type**.
* Keep in mind that **each type of diver** has a **different level of oxygen** and it willimplement the **method differently**.

#### hit(fish: BaseFish)

* The method takes a fish parameter. This fish represents the target fish that the diver is trying to catch. The method performs the following actions:
* The diver's **oxygen\_level** is reduced by the duration specified by the **time\_to\_catch** property of the given fish. It's important to note that the **oxygen level should not drop below 0**. If the diver's oxygen level is less than the time required to catch the fish, you should set the diver's **oxygen\_level** to **0**.
* Only if the diver's **oxygen\_level** is sufficient to catch the **fish** should it be added to the diver's **catch** list.
* If the fish is caught, the diver's **competition\_points** increase by the value of the **points** property of the caught **fish**, rounded to one decimal place.

#### update\_health\_status()

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

**\_\_str\_\_()**

* Returns a **string** with **information** about the **Diver** in the format below.

**"{type of diver}: [Name: {name}, Oxygen level left: {oxygen\_level}, Fish caught: {count of caught fish}, Points earned: {competition\_points}]"**

## Class FreeDiver

In the **free\_diver.py** file, the class **FreeDiver** should be implemented. The free diver is a **type of diver**. Each diver has an **initial oxygen level value of 120**.

### Methods

#### \_\_init\_\_(name: str)

* In the **\_\_init\_\_** method, all the needed attributes must be set.

#### miss(time\_to\_catch: int)

* This method reduces the diver's **oxygen\_level** by **60% of the time\_to\_catch** value of the missed fish. If the calculated value **is not a whole number, it is rounded to the nearest whole integer**. Additionally, it ensures that the oxygen level does not fall **below 0**. If the diver's **current oxygen level** is insufficient to catch the fish, the method sets the diver's **oxygen\_level** to **0**, preventing it from going into negative values.

#### renew\_oxy()

* Restoring the **oxygen\_level** to its original value for divers of this type.

## Class ScubaDiver

In the **scuba\_diver.py** file, the class **ScubaDiver** should be implemented. The scuba diver is a **type of diver**. Each diver has an **initial oxygen level value of 540**.

### Methods

#### \_\_init\_\_(name: str)

* In the **\_\_init\_\_** method, all the needed attributes must be set.

#### miss(time\_to\_catch: int)

* This method reduces the diver's **oxygen\_level** by **30% of of the time\_to\_catch** value of the missed fish. If the calculated value **is not a whole number, it is rounded to the nearest whole integer**. Additionally, it ensures that the oxygen level does not fall **below 0**. If the diver's **current oxygen level** is insufficient to catch the fish, the method sets the diver's **oxygen\_level** to **0**, preventing it from going into negative values.

#### renew\_oxy()

* Restoring the **oxygen\_level** to its original value for divers of this type.

## Class NauticalCatchChallengeApp

In the **nautical\_catch\_challenge\_app.py** file, the class **NauticalCatchChallengeApp** should be implemented. It will contain the functionality of the project.

### Structure

The class should have the following attributes:

* **divers: list**
  + An empty list to store all diver objects assigned for the competition.
* **fish\_list: list**
  + An empty list for storing all fish objects that are allowed for chasing in the competition.

### Methods

#### \_\_init\_\_()

* In the **\_\_init\_\_** method, all the needed attributes must be set.

#### dive\_into\_competition(diver\_type: str, diver\_name: str)

The method **creates** a diver of the given type and **adds** it to the **divers** collection.

* If the diver’s type is **not valid**, return the following message:

**"{diver\_type} is not allowed in our competition."**

* If a diver with the same **name** is already added to the list, **do not duplicate records**, return the following message:

**"{diver\_name} is already a participant."**

* If none of the above cases is reached, the **diver** is successfully created. Store the diver in the appropriate collection and return it:

**"{diver\_name} is successfully registered for the competition as a {diver\_type}."**

* **Valid types** of divers are: **"FreeDiver"** and **"ScubaDiver"**

#### swim\_into\_competition(fish\_type: str, fish\_name: str, points: float)

The method **creates** a fish of the given type and **adds** them to the **fish** collection. The method is responsible for **allowing a new fish to chase** into the competition.

* First, check if the **fish type** is valid, and if **not**, return the following message:

**"{fish\_type} is forbidden for chasing in our competition."**

* Then, check if the fish **name** is already added to the list, **do not duplicate records**, and return the following message:

**"{fish\_name} is already permitted."**

* If the above case is not reached, create the correct type of **fish** and add it to the appropriate collection. Return the following message:

**"{fish\_name} is allowed for chasing as a {fish\_type}."**

* **Valid types** of fish are: **"PredatoryFish"** and **"DeepSeaFish"**.

#### chase\_fish(diver\_name: str, fish\_name: str, is\_lucky: bool)

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

* **Diver Validation:**
  + Validates the existence of a diver with the given **diver\_name** in the collection of registered **divers**.
  + If **no diver is found**, the method returns the message:

**"{diver\_name} is not registered for the competition."**

* **Fish Validation:**
  + Validates the existence of a fish with the given **fish\_name** in the **list of allowed fish**.
  + If **no fish is found**, the method returns the message:

**"The {fish\_name} is not allowed to be caught in this competition."**

* **Health Check:**
  + Checks if the diver has a **has\_health\_issue** equal to **True** then returns the message:

**"{diver\_name} will not be allowed to dive, due to health issues."**

* **Oxygen Level Comparison:**
  + If the diver’s **oxygen\_level** is less than the fish's **time\_to\_catch** value, the fish escapes, the **diver misses** with the harpoon, and the method returns:

**"{diver\_name} missed a good {fish\_name}."**

* + If the diver's **oxygen\_level** is equal to the fish's **time\_to\_catch**:
* If **is\_lucky** is **True**, the diver successfully catches the fish by invoking the **hit** method with the targeted fish and return the relevant message:

**"{diver\_name} hits a {points}pt. {fish\_name}."**

* If **is\_lucky** is **False**, the diver misses the fish by invoking the **miss** method and return the relevant message:

**"{diver\_name} missed a good {fish\_name}."**

* If the diver’s **oxygen\_level** is **greater than** the fish's **time\_to\_catch** value, the fish is caught, and the diver **hits** with the harpoon. The method returns:

**"{diver\_name} hits a {points}pt. {fish\_name}."**

* **Zero Oxygen Level Handling:**
* If, at any point during the chase, the diver's **oxygen\_level drops to 0**, the diver **has\_health\_issue** property is set to **True**.

#### health\_recovery()

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 **has\_health\_issue** property set to **True**, it initiates a series of actions to stabilize the diver:

* First, it sets the **has\_health\_issue** property of the diver to **False**, indicating that the diver is now in a stable condition.
* Secondly, it replenishes the diver's **oxygen\_level** **back to its maximum**, ensuring the divers are ready for the next dive when they feel comfortable.
* Returnsthe following message:

**"Divers recovered: {count}**"

#### diver\_catch\_report(diver\_name: str)

Returns detailed information about **a specific diver and his catch so far:**

"\*\*{diver\_name} Catch Report\*\*

**{fish details1}**

**{fish details2}**

**…**

**{fish detailsn}"**

**competition\_statistics()**

Return information about each diver, arranging them in **descending order** based on **competition\_points**. If more than one diver has the same number of points, further arrange them in **descending order** based on the **count of** catches. For divers with the same catch count, arrange them **alphabetically** by name. Return **only divers that are in good health condition**. To receive the correct output, use the \_\_str\_\_()method **of each diver:**

**"\*\*Nautical Catch Challenge Statistics\*\***

**{diver1}**

**{diver2}**

**...**

**{divern}"**

#### Examples

|  |
| --- |
| **Input** |
| **nautical\_catch\_challenge = NauticalCatchChallengeApp()**  *# Dive into competition*  **print(nautical\_catch\_challenge.dive\_into\_competition("ScubaDiver", "MaxineHarper"))**  **print(nautical\_catch\_challenge.dive\_into\_competition("FreeDiver", "JamalCarter"))**  **print(nautical\_catch\_challenge.dive\_into\_competition("SkyDiver", "FionaBennett"))**  **print(nautical\_catch\_challenge.dive\_into\_competition("FreeDiver", "OscarWallace"))**  **print(nautical\_catch\_challenge.dive\_into\_competition("ScubaDiver", "LilaMoreno"))**  **print(nautical\_catch\_challenge.dive\_into\_competition("FreeDiver", "LilaMoreno"))**  *# Swim into competition*  **print(nautical\_catch\_challenge.swim\_into\_competition("ReefFish", "AzureDamselfish", 8.7))**  **print(nautical\_catch\_challenge.swim\_into\_competition("DeepSeaFish", "BluestripeSnapper", 6.3))**  **print(nautical\_catch\_challenge.swim\_into\_competition("PredatoryFish", "YellowtailSurgeonfish", 5.0))**  **print(nautical\_catch\_challenge.swim\_into\_competition("PredatoryFish", "Barracuda", 9.2))**  **print(nautical\_catch\_challenge.swim\_into\_competition("PredatoryFish", "Coryphaena", 9.7))**  **print(nautical\_catch\_challenge.swim\_into\_competition("PredatoryFish", "Bluefish", 4.4))**  **print(nautical\_catch\_challenge.swim\_into\_competition("DeepSeaFish", "SwordFish", 10.0))**  **print(nautical\_catch\_challenge.swim\_into\_competition("DeepSeaFish", "Mahi-Mahi", 9.1))**  **print(nautical\_catch\_challenge.swim\_into\_competition("DeepSeaFish", "Tuna", 8.5))**  **print(nautical\_catch\_challenge.swim\_into\_competition("AquariumFish", "SilverArowana", 3.3))**  **print(nautical\_catch\_challenge.swim\_into\_competition("DeepSeaFish", "Barracuda", 8.6))**  *# Conduct fishing competitions*  **print(nautical\_catch\_challenge.chase\_fish("FionaBennett", "AzureDamselfish", False))**  **print(nautical\_catch\_challenge.chase\_fish("JamalCarter", "SilverArowana", True))**  **print(nautical\_catch\_challenge.chase\_fish("MaxineHarper", "YellowtailSurgeonfish", False))**  **print(nautical\_catch\_challenge.chase\_fish("MaxineHarper", "Mahi-Mahi", False))**  **print(nautical\_catch\_challenge.chase\_fish("MaxineHarper", "Tuna", False))**  **print(nautical\_catch\_challenge.chase\_fish("MaxineHarper", "Coryphaena", True))**  **print(nautical\_catch\_challenge.chase\_fish("MaxineHarper", "BluestripeSnapper", True))**  **print(nautical\_catch\_challenge.chase\_fish("OscarWallace", "Barracuda", False))**  **print(nautical\_catch\_challenge.chase\_fish("OscarWallace", "YellowtailSurgeonfish", False))**  **print(nautical\_catch\_challenge.chase\_fish("OscarWallace", "Tuna", True))**  **print(nautical\_catch\_challenge.chase\_fish("JamalCarter", "Barracuda", True))**  **print(nautical\_catch\_challenge.chase\_fish("JamalCarter", "YellowtailSurgeonfish", True))**  **print(nautical\_catch\_challenge.chase\_fish("LilaMoreno", "Tuna", False))**  **print(nautical\_catch\_challenge.chase\_fish("LilaMoreno", "Mahi-Mahi", False))**  **print(nautical\_catch\_challenge.chase\_fish("LilaMoreno", "SwordFish", True))**  *# Check health recovery*  **print(nautical\_catch\_challenge.health\_recovery())**  *# Conduct fishing competitions*  **print(nautical\_catch\_challenge.chase\_fish("LilaMoreno", "Tuna", False))**  **print(nautical\_catch\_challenge.chase\_fish("LilaMoreno", "Mahi-Mahi", False))**  **print(nautical\_catch\_challenge.chase\_fish("LilaMoreno", "SwordFish", True))**  *# View catch reports*  **print(nautical\_catch\_challenge.diver\_catch\_report("LilaMoreno"))**  *# View competition statistics*  **print(nautical\_catch\_challenge.competition\_statistics())** |

|  |
| --- |
| **Output** |
| **MaxineHarper is successfully registered for the competition as a ScubaDiver.**  **JamalCarter is successfully registered for the competition as a FreeDiver.**  **SkyDiver is not allowed in our competition.**  **OscarWallace is successfully registered for the competition as a FreeDiver.**  **LilaMoreno is successfully registered for the competition as a ScubaDiver.**  **LilaMoreno is already a participant.**  **ReefFish is forbidden for chasing in our competition.**  **BluestripeSnapper is allowed for chasing as a DeepSeaFish.**  **YellowtailSurgeonfish is allowed for chasing as a PredatoryFish.**  **Barracuda is allowed for chasing as a PredatoryFish.**  **Coryphaena is allowed for chasing as a PredatoryFish.**  **Bluefish is allowed for chasing as a PredatoryFish.**  **SwordFish is allowed for chasing as a DeepSeaFish.**  **Mahi-Mahi is allowed for chasing as a DeepSeaFish.**  **Tuna is allowed for chasing as a DeepSeaFish.**  **AquariumFish is forbidden for chasing in our competition.**  **Barracuda is already permitted.**  **FionaBennett is not registered for the competition.**  **The SilverArowana is not allowed to be caught in this competition.**  **MaxineHarper hits a 5.0pt. YellowtailSurgeonfish.**  **MaxineHarper hits a 9.1pt. Mahi-Mahi.**  **MaxineHarper hits a 8.5pt. Tuna.**  **MaxineHarper hits a 9.7pt. Coryphaena.**  **MaxineHarper will not be allowed to dive, due to health issues.**  **OscarWallace hits a 9.2pt. Barracuda.**  **OscarWallace missed a good YellowtailSurgeonfish.**  **OscarWallace will not be allowed to dive, due to health issues.**  **JamalCarter hits a 9.2pt. Barracuda.**  **JamalCarter missed a good YellowtailSurgeonfish.**  **LilaMoreno hits a 8.5pt. Tuna.**  **LilaMoreno hits a 9.1pt. Mahi-Mahi.**  **LilaMoreno hits a 10.0pt. SwordFish.**  **Divers recovered: 4**  **LilaMoreno hits a 8.5pt. Tuna.**  **LilaMoreno hits a 9.1pt. Mahi-Mahi.**  **LilaMoreno hits a 10.0pt. SwordFish.**  **\*\*LilaMoreno 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: 10.0, 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: 10.0, Time to Catch: 180 seconds]**  **\*\*Nautical Catch Challenge Statistics\*\***  **ScubaDiver: [Name: MaxineHarper, Oxygen level left: 540, Fish caught: 4, Points earned: 32.3]**  **FreeDiver: [Name: JamalCarter, Oxygen level left: 120, Fish caught: 1, Points earned: 9.2]**  **FreeDiver: [Name: OscarWallace, Oxygen level left: 120, Fish caught: 1, Points earned: 9.2]** |

# Task 3: Unit Tests (100 points)

You will **be provided with another skeleton** for this problem. **Open** the **new skeleton** as a **new project** and write tests for the **RailwayStation** class. The class will have some methods, fields, and one constructor, all of them working properly. You are **NOT ALLOWED** to change anything in the class code. Cover the whole class with unit tests to make sure that the class is working as intended. Submit **only the test** folder.