# Python OOP Retake Exam - 15 August 2023



*Handball is a team sport played with a ball, where two teams of seven players (including a goalkeeper) compete to score the most goals during the game. The gameplay combines rapid attacks and strategic defense. We are developing a support application to enhance understanding and practice of the game, assisting teams with information and protection for effective play.*

You will be provided with a **skeleton** that includes all the folders and files that you will need.

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

A screenshot of a computer

Description automatically generated with medium confidence

**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 **in the zip file** your **venv**, **.idea**, **pycache**, and **\_\_MACOSX** (for Mac users), 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 the **structure and functionality** of all the classes (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 BaseEquipment

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

### Structure

The class should have the following attributes:

* **protection:** int
  + The value represents the **protection of the equipment**.
* **price:** float
  + The value represents the **price of the equipment**.

### Methods

#### \_\_init\_\_(protection: int, price: float)

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

#### increase\_price()

* Method **increases the equipment’s price**. Keep in mind that **each type of equipment** implements the method **differently**.

## Class KneePad

In the **knee\_pad.py** file, the class **KneePad** should be implemented. The knee pad is a **type of equipment**. Each knee pad equipment has a **protection of 120 and a price of 15.0 EUR**.

### Methods

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

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

#### increase\_price()

* The method **increases** the **price** by **20% (twenty percent)**.

## Class ElbowPad

In the **elbow\_pad.py** file, the class **ElbowPad** should be implemented. An elbow pad is a **type of equipment**. Each elbow pad has a **protection of 90 and a price of 25.0 EUR**.

### Methods

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

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

#### increase\_price()

* The method **increases** the **price** by **10% (ten percent)**.

## Class BaseTeam

In the **base\_team.py** file, the class **BaseTeam** should be implemented. It is a **base class** for any **type of team,** 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 team**.
  + If the name is **an empty string or contains only white spaces**, raise a ValueError with the message: **"Team name cannot be empty!"**
* **country:** str
  + The value represents the **country of origin** **of a team**. It should be **at least 2 symbols long (no leading or trailing white spaces counts)**.
  + If the team’s country **is less than 2 symbols long**, raise a ValueError with the message: **"Team country should be at least 2 symbols long!"**
* **advantage:** int
  + The value represents the **advantage in points** that **each team** has.
  + If the team’s advantage **is less than or equal to 0**, raise a ValueError with the message: **"Advantage must be greater than zero!"**
* **budget: float**
  + **The value represents the team’s budget.**
* **wins: int**
  + **The value represents the team’s wins, initially set to 0.**
* **equipment:** list
  + Empty list that **will contain equipment(objects)** each team has.

### Methods

#### \_\_init\_\_(name: str, country: str, advantage: int, budget: float)

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

#### win()

* **Increases** the **team’s advantage** and the **number of wins.** Keep in mind that **each type of team** implements the method **differently**.

#### get\_statistics()

The method **returns** the statistics about the team in the following format, **each line on a new row**:

**"Name: {team\_name}**

**Country: {team\_country}**

**Advantage: {team\_advantage} points**

**Budget: {team\_budget}EUR**

**Wins: {number\_of\_wins}**

**Total Equipment Price: {total\_price\_of\_team\_equipment}**

**Average Protection: {avg\_team\_protection}"**

* The **budget** and the **total equipment price** should be **formatted** to the **second decimal places**.
* **Average Protection** refers to the property **protection** of each piece of **equipment** that the team has in its **equipment collection**. **Round** the **average protection** to the **smaller integer**.

## Class OutdoorTeam

In the **outdoor\_team.py** file, the class **OutdoorTeam** should be implemented. The outdoor team is a **type of team**. Each outdoor team has an **initial budget of 1000.0 EUR**.

### Methods

#### \_\_init\_\_(name: str, country: str, advantage: int)

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

#### win()

* The method **increases** the team’s **advantage** **by 115 points**. Remember to increase the **wins number** as well.

## Class IndoorTeam

In the **indoor\_team.py** file, the class **IndoorTeam** should be implemented. The indoor team is a **type of team**. Each indoor team has an **initial budget of 500.0 EUR**.

### Methods

#### \_\_init\_\_(name: str, country: str, advantage: int)

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

#### win()

* The method **increases** the team’s **advantage** **by 145 points**. Remember to increase the **wins number** as well.

## Class Tournament

In the **tournament.py** file, the class **Tournament** should be implemented. It will contain the functionality of the project.

### Structure

The class should have the following attributes:

* **name:** str
  + The value represents the **name of the tournament**.
  + The **name** should contain **letters and digits only**. If the name has other symbols, raise a ValueError with the message: **"Tournament name should contain letters and digits only!"**
* **capacity: int**
  + The **number** of **teams** а **Tournament** **can have.**
* **equipment: list**
  + Empty list that **will contain all equipment** (objects) that are created.
* **teams: list**
  + Empty list that **will contain all teams** (objects) that are created.

### Methods

#### \_\_init\_\_(name: str, capacity: int)

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

#### add\_equipment(equipment\_type: str)

The method **creates** equipment of the given type and **adds** it to the **equipment** collection.

* If the equipment’s type is not valid, raise an **Exception** with the following message:

**"Invalid equipment type!"**

* Otherwise, **create** the equipment, **add** it to the equipment list, and **return** the following message:

**"{equipment\_type} was successfully added."**

* **Valid types** of equipment are: **"KneePad"** and **"ElbowPad"**

#### add\_team(team\_type: str, team\_name: str, country: str, advantage: int)

The method **creates** a team of the given type and **adds** it to the **teams’** collection.   
All teams’ **names** will be **unique**.

* **First**, check if the **team type** is valid, and if **not** raise an **Exception** with the following message:

**"Invalid team type!"**

* **Then**, check if there is an available **tournament** **capacity**,and ifnot **return** the following message:

**"Not enough tournament capacity."**

* Otherwise, **create** the team, **add** it to the **teams’ list**, and **return** the following message:

**"{team\_type} was successfully added."**

* **Valid types** of teams are: **"OutdoorTeam"** and **"IndoorTeam"**.

#### sell\_equipment(equipment\_type: str, team\_name: str)

The method **adds the equipment** of the given type to the **team’s equipment** collection. Both **equipment** and **team** will **always exist**.

* **First**, check if the equipment **can be sold** to the team. If the team’s **budget** is **not enough** to buy the equipment, **raise an Exception** with the following message:

**"Budget is not enough!"**

* If the equipment can be **sold** to the team, **remove** it from the **tournament's equipment collection**, and **add it** to the **team’s equipment collection**. Decrease the **budget** with the **equipment’s price**. **Return** the following message:

**"Successfully sold {equipment\_type} to {team\_name}."**

* Take the **last equipment** of the given **type** from the collection.

#### remove\_team(team\_name: str)

The method **removes the team** with the given **name** from the **tournament**.

* **First**, check if there is a team with the given **name** in the **team’s collection**. If not, **raise an Exception** with thefollowing message:

**"No such team!"**

* **Then**, check if the team **has any wins.** If so, **raise an Exception** with thefollowing message:

**"The team has {number\_of\_wins} wins! Removal is impossible!"**

* If the team can be **removed successfully**, **remove** it from the tournament, and **return** the following message: **"Successfully removed {team\_name}."**

#### increase\_equipment\_price(equipment\_type: str)

The method **increases** the **price** for **all equipment** of the **given type** that is in the **tournament’s equipment collection**. The equipment type will be one of the **valid types** (**KneePad** or **ElbowPad**). When all prices for the given equipment type are successfully changed (**hint**: use increase\_price() method), **return** the following message:

**"Successfully changed {number\_of\_changed\_equipment}pcs of equipment."**

* Equipment that is **already sold to teams** should **not** **be affected**.

#### play(team\_name1: str, team\_name2: str)

The method **starts a game** between **two teams**. The **team’s names** **will always exist and will be unique**.

* **First**, check if **both teams** are from **one and the same type** (**IndoorTeam or OutdoorTeam**). If not, **raise an Exception** with thefollowing message:

**"Game cannot start! Team types mismatch!"**

* **Then**, **sum** the **points of advantage** and the **total protection** for **each team**. You will need to **compare** the **results**:
  + The **team** with the **greater result (the sum of advantage points and total protection) wins**. You have to **increase** the **winner’s** points of **advantage** and the number of **wins.** You can use the team’s **win()** method. **Return** the following message:

**"The winner is {team\_name\_of\_winner}."**

* + In case the **teams** happen to be with **equal results**, **return** the following message:

**"No winner in this game."**

#### get\_statistics()

Returns information about the **tournament** and the **teams in the tournament, sorted by number of wins, descending**. Each on a **new line**. Use the team’s **get\_statistics()** method.

"Tournament: {tournament\_name}

Number of Teams: {number\_of\_teams}

Teams:

{team1\_statistics}

{team2\_statistics}

…

{teamN\_statistics}"

#### Examples

|  |
| --- |
| **Input** |
| **t = Tournament('SoftUniada2023', 2)**  **print(t.add\_equipment('KneePad'))**  **print(t.add\_equipment('ElbowPad'))**  **print(t.add\_team('OutdoorTeam', 'Levski', 'BG', 250))**  **print(t.add\_team('OutdoorTeam', 'Spartak', 'BG', 250))**  **print(t.add\_team('IndoorTeam', 'Dobrich', 'BG', 280))**  **print(t.sell\_equipment('KneePad', 'Spartak'))**  **print(t.remove\_team('Levski'))**  **print(t.add\_team('OutdoorTeam', 'Lokomotiv', 'BG', 250))**  **print(t.increase\_equipment\_price('ElbowPad'))**  **print(t.increase\_equipment\_price('KneePad'))**  **print(t.play('Lokomotiv', 'Spartak'))**  **print(t.get\_statistics())** |
| **Output** |
| **KneePad was successfully added.**  **ElbowPad was successfully added.**  **OutdoorTeam was successfully added.**  **OutdoorTeam was successfully added.**  **Not enough tournament capacity.**  **Successfully sold KneePad to Spartak.**  **Successfully removed Levski.**  **OutdoorTeam was successfully added.**  **Successfully changed 1pcs of equipment.**  **Successfully changed 0pcs of equipment.**  **The winner is Spartak.**  **Tournament: SoftUniada2023**  **Number of Teams: 2**  **Teams:**  **Name: Spartak**  **Country: BG**  **Advantage: 365 points**  **Budget: 985.00EUR**  **Wins: 1**  **Total Equipment Price: 15.00**  **Average Protection: 120**  **Name: Lokomotiv**  **Country: BG**  **Advantage: 250 points**  **Budget: 1000.00EUR**  **Wins: 0**  **Total Equipment Price: 0.00**  **Average Protection: 0** |

# 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 **Trip** 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.