



Project Title:

Table Tennis Equipment

Project Description:

Table Tennis is a game played inside by two or four people. It requires only two player paddles, lightweight balls, and a normal table to play its fast-paced rallies. The code I have below as a requirement for OOPs final project demonstrates an Object-Oriented approach in representing table tennis equipment. It utilizes inheritance and abstraction to create a hierarchy of classes that model different equipment items and their properties. It also touches upon another core OOP concept the Encapsulation and Polymorphism.

Objectives:

The objective of this code is to model Table Tennis equipment using an Object-Oriented Programming approach. This approach aims to:

- Promote code reusability by establishing a base class (Equipment) that captures common features (brand, name) applicable to various equipment types, minimizing redundancy.
- Encourage maintainability by allowing subclasses like Paddle and Table to inherit core functionalities from Equipment while still enabling customization for their specific behaviors. This simplifies future additions or changes to equipment types.
- Enhances code readability by mirroring real-world equipment (Paddle, Table) with their associated characteristics and functionalities, making it easier to understand for both developers and non-developers unfamiliar with the initial implementation.



Importance and Contribution of the Project:

This project's importance lies in two key areas. First, it showcases the main object-oriented programming (OOP) principles: abstraction, encapsulation, inheritance, and polymorphism. These principles contribute to well-structured and maintainable code. Second, this project serves as a foundation for creating a larger OOP project related to table tennis. You can expand on it by adding new equipment classes and functionalities, such as implementing cost calculation, stock management, or game simulation.

Anyone interested in understanding table tennis equipment can benefit from this code. The code serves as a basic reference for how Table Tennis equipment can be categorized and represented using object-oriented programming (OOP) concepts. It clearly separates common attributes (brand, name) from specific details (grip style, material). This separation makes the code easier to understand and maintain.

Four Principles of Object-Oriented Programming:

Inheritance:

```
class TableTennisItem(Equipment):
```

- *TableTennisItem* inherits from *Equipment*. This means *TableTennisItem* inherits all the attributes and methods defined in *Equipment*.

```
class Paddle(TableTennisItem):
```

```
class Table(TableTennisItem):
```

- *Paddle* and *Table* inherit from *TableTennisItem*. They gain access to the attributes and methods of both *Equipment* and *TableTennisItem*.

Polymorphism:

```
class Equipment(ABC):  
    ...  
    @abstractmethod  
    def describe(self):  
        pass
```

- The abstract *describe* method in *Equipment* forces subclasses to implement their own version.

```
class TableTennisItem(Equipment):  
    def describe(self):  
        ...
```

```
class Paddle(TableTennisItem):  
    ...  
    def describe(self):  
        return f"{super().describe()} Paddle - {self.describe_grip()}"
```

```
class Table(TableTennisItem):  
    ...  
    def describe(self):  
        return f"{super().describe()} Table - {self.describe_material()}"
```

- *TableTennisItem*, *Paddle*, and *Table* all override the *describe* method to provide specific descriptions for their respective equipment types.

Encapsulation:

```
self.__brand = brand  
self.__name = name
```

- The *Equipment* class uses private attributes, the double underscores before the attribute names. This prevents direct access from outside the class, promoting data protection.

```
def get_brand(self):
```

```
def get_name(self):
```

- Define getter methods to access the private attributes. These methods control how the data is retrieved, potentially adding validation or formatting logic.

Abstract:

```
@abstractmethod
def describe(self):
    pass
```

- @abstractmethod decorator: This tells that the method describe is abstract. An abstract method is one that is not specifically implemented in the class itself.
- def describe(self):: This defines the method signature, specifying its name (describe) and that it takes the object itself (self) as an argument.
- pass: This is a placeholder within the method. Because it is abstract, it does not need an actual implementation in the body of the Equipment class.

By designating describe as an abstract method, the class requires its subclasses to implement a description method to provide a "specific description for each item". This is to make sure that all the objects of the Equipment class would describe themselves

Hardware and Software Used:

Hardware:

- Computer
- Smartphone

Software:

- OnlineGDB



Output:

```
Butterfly Nittaku Ball: Butterfly Nittaku Ball
Yasaka Ma Lin Pro Paddle: Yasaka Ma Lin Pro Paddle - This paddle is designed for a Shakehand grip.
Cornilleau Sport 1000 Outdoor Table: Cornilleau Sport 1000 Outdoor Table - This table is made of Melamine.
```

Description:

In the Butterfly Nittaku Ball this is the simplest output, showing just the brand and name ("Butterfly Nittaku Ball") because it uses the base implementation of describe from



TableTennisItem. In Yasaka Ma Lin Pro Paddle this output is more detailed, including the brand, name, and grip style ("Yasaka Ma Lin Pro Paddle - This paddle is designed for a Shakehand grip"). This is achieved through overriding describe in Paddle to combine the base description with grip information. And lastly, in the Cornilleau Sport 1000 Outdoor Table, it is similar to the paddle, this output provides brand, name, and additional details ("Cornilleau Sport 1000 Outdoor Table - This table is made of Melamine"). Overriding describe in Table allows for including the material information.

Codes:

```
from abc import ABC, abstractmethod

class Equipment(ABC):
    def __init__(self, brand, name):
        self.__brand = brand    # Make brand private
        self.__name = name      # Make name private

    def get_brand(self):
        return self.__brand

    def get_name(self):
        return self.__name

    @abstractmethod
    def describe(self):
        pass

class TableTennisItem(Equipment):
    def describe(self):
        return f"{self.get_brand()} {self.get_name()}"

class Paddle(TableTennisItem):
    def __init__(self, brand, name, grip_style):
        super().__init__(brand, name)
        self.grip_style = grip_style

    def describe_grip(self):
        return f"This paddle is designed for a {self.grip_style} grip."

    def describe(self):
        return f"{super().describe()} Paddle - {self.describe_grip()}"
```



```
class Table(TableTennisItem):
    def __init__(self, brand, name, material):
        super().__init__(brand, name)
        self.material = material

    def describe_material(self):
        return f"This table is made of {self.material}."

    def describe(self):
        return f"{super().describe()} Table - {self.describe_material()}"

# Creating objects
butterfly_ball = TableTennisItem("Butterfly", "Nittaku Ball")
yasaka_paddle = Paddle("Yasaka", "Ma Lin Pro", "Shakehand")
cornilleau_table = Table("Cornilleau", "Sport 1000 Outdoor", "Melamine")

# Accessing object attributes and methods
print("Butterfly Nittaku Ball:", butterfly_ball.describe())

print("Yasaka Ma Lin Pro Paddle:", yasaka_paddle.describe())

print("Cornilleau Sport 1000 Outdoor Table:", cornilleau_table.describe())
```

Code link: <https://onlinegdb.com/O5oOJPqSwh>

User Guide:

This guide helps you use the Python code to manage information about table tennis equipment.

The Code Offers:

- Classes representing equipment:
 - Equipment (Abstract): Base class for all equipment (brand, name).
 - TableTennisItem: Generic table tennis item (inherits from Equipment).
 - Paddle and Table: Specific equipment types with unique attributes (grip style for paddle, material for table).



Using the Code:

1. Import the Classes:

```
from abc import ABC, abstractmethod  
  
# Classes are defined here (see original code)
```

2. Create Equipment Objects:

Use the appropriate class with its specific arguments (brand, name, and additional details).

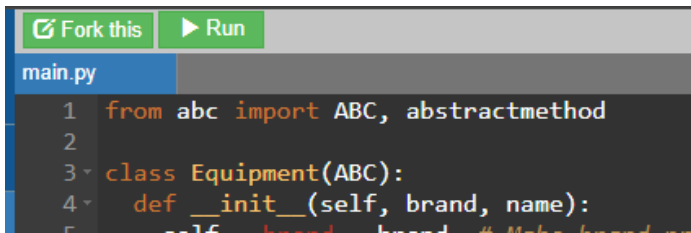
```
# Table tennis ball  
butterfly_ball = TableTennisItem("Butterfly", "Nittaku Ball")  
  
# Yasaka paddle with Shakehand grip  
yasaka_paddle = Paddle("Yasaka", "Ma Lin Pro", "Shakehand")  
  
# Cornilleau table made of Melamine  
cornilleau_table = Table("Cornilleau", "Sport 1000 Outdoor", "Melami
```

3. Get Information:

- Use `get_brand` and `get_name` (inherited from `Equipment`) to retrieve brand and name of any equipment object.
- Utilize the specific `describe` method of each class for a detailed description.

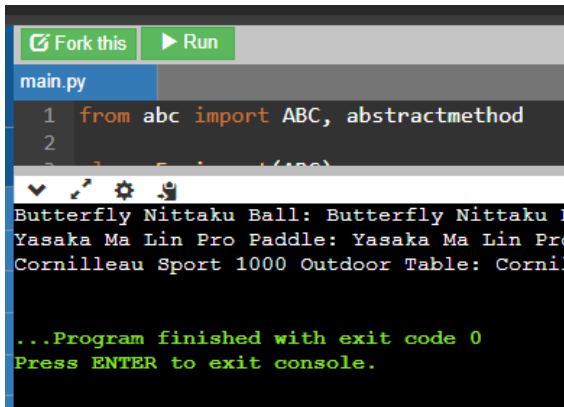
```
print(butterfly_ball.describe()) # "Butterfly Nittaku Ball"  
print(yasaka_paddle.describe()) # "Specific description with grip"  
print(cornilleau_table.describe()) # "Specific description with ma
```

4. After everything, if you wanted to run the code (onlineGDB in the example):



```
Fork this Run
main.py
1 from abc import ABC, abstractmethod
2
3 class Equipment(ABC):
4     def __init__(self, brand, name):
5         self.brand = brand # Make brand an
```

- Click "Run"



```
Fork this Run
main.py
1 from abc import ABC, abstractmethod
2
3 class Equipment(ABC):
4     def __init__(self, brand, name):
5         self.brand = brand # Make brand an

Butterfly Nittaku Ball: Butterfly Nittaku B
Yasaka Ma Lin Pro Paddle: Yasaka Ma Lin Pro
Cornilleau Sport 1000 Outdoor Table: Cornil

...Program finished with exit code 0
Press ENTER to exit console.
```

- Then you'll see the output below.

References:

- The built-in abc module used for abstract classes:
<https://docs.python.org/3/library/abc.html>
- Inheritance concepts: https://www.w3schools.com/python/python_inheritance.asp
- Programiz - Polymorphism in Python: <https://www.programiz.com/python-programming/polymorphism>
- W3Schools - Python Polymorphism:
https://www.w3schools.com/python/python_polymorphism.asp
- GeeksforGeeks - Polymorphism in Python: <https://www.geeksforgeeks.org/polymorphism-in-python/>
- TutorialPoint – Python Encapsulation:
https://www.tutorialspoint.com/python/python_encapsulation.htm