Class Design Strategy In Python

Boat Earphone Class

```
------------------------------------------------------------------------------------------------------------------------
"""
    @Author:Kunal Narkhede
    @Date:27/12/2023
    @Goal:To implement class Earphone
    Capture Real Life Product on Amazon
    http://surl.li/oqujf

 """
------------------------------------------------------------------------------------------------------------------------
import sys
class ProductDimension:
    """
        This class implement the Dimension of Earphone
        @__init__(self, length: float, width: float, height: float, weight: float):
        Constructor
        @get_length(self)
            getter of attribute length
        @get_width(self)
            getter of attribute width
        @get_height(self)
            getter of attribute height
        @get_weight(self)
            getter of attribute weight
        -------------------------------------
        @set_length(self):
            setter of attribute length
        @set_width(self):
            setter of attribute width
        @set_height(self):
            setter of attribute height
        @set_weight(self):
            setter of attribute weight
    """
    def __init__(
             self,
             length:float,
             width:float,
             height:float,
             weight:float
        ):

        """
            Constructor of ProductDimension class:
            @__init__(self, length: float, width: float, height: float, weight: float):

            @self:newly created class object of ProductDimension
            @length:Client specified value for attribute length
            @width:Client specified value for attribute width
            @height:Client specified value for attribute height
            @weight:Client specified value for attribute weight

        """
        if type(length)!=float:
            raise TypeError("Bad type:length")
        if type(width)!=float:
            raise TypeError("Bad type:width")
        if type(height)!=float:
            raise TypeError("Bad type:height")
        if type(weight)!=float:
            raise TypeError("Bad type:weight")
        if length<=0.0:
            raise ValueError("Length must be positive")
        if width<=0.0:
            raise ValueError("Width must be positive")
        if height<=0.0:
            raise ValueError("Height must be positive")
        if weight<=0.0:
            raise ValueError("Weight must be positive")

        self.length=length
        self.width=width
        self.height=height
        self.weight=weight

     #getter method

    def get_length(self) -> float:
        """
            Returns the length attribute of the calling object
        """
        return self.length

    def get_width(self) -> float:
        """
            Returns the width attribute of the calling object
        """
        return self.width

    def get_height(self) -> float:
        """
            Returns the height attribute of the calling object
        """
        return self.height

    def get_weight(self) -> float:
        """
            Returns the weight attribute of the calling object
        """
        return self.weight

    #setter method
```

```python

    def set_length(self,new_length:float):
        """
            Sets the length attribute of the calling object to @new_length
            Before setting, TypeCheck and ValueCheck is performed.
        """
        if type(new_length)!=float:
            raise TypeError("new_length must be an float")
        if new_length <= 0.0:
            raise TypeError("new_length must be positive")
        self.length=new_length

    def set_width(self,new_width:float):
        """
            Sets the width attribute of the calling object to @new_width
            Before setting, TypeCheck and ValueCheck is performed.
        """
        if type(new_width)!=float:
            raise TypeError("new_width must be an float")
        if new_width <= 0.0:
            raise ValueError("new_width must be positive")
        self.width=new_width

    def set_height(self,new_height:float):
        """
            Sets the height attribute of the calling object to @new_height
            Before setting, TypeCheck and ValueCheck is performed.
        """
        if type(new_height)!=float:
            raise TypeError("new_height must be an float")
        if new_height <= 0.0 :
            raise ValueError("new_height must be positive")
        self.height=new_height

    def set_weight(self,new_weight:float):
        """
           Sets the weight attribute of the calling object to @new_weight
           Before setting, TypeCheck and ValueCheck is performed.
        """
        if type(new_weight)!=float:
           raise TypeError("new_weight must be an float")
        if new_weight <= 0.0:
           raise ValueError("new_weight must be positive")
        self.weight=new_weight


class Earphone:
    def __init__(self,
              er_brand:str,
              er_cable_feature:str,
              er_net_quantity_in_piece:int,
              er_colour:str,
              er_battery_backup_in_hrs:int,
              er_control_method:str,
              er_nr_of_items:int,
              er_manufacturer:str,
              er_model:str,
              er_prod_dimensions:ProductDimension,
              er_batteries:str,
              er_model_nr:str,
              er_special_feature:str,
              er_mounting_hardware:[str],
              er_batteries_included:bool,
              er_batteries_required:bool,
              er_batteries_cell_composition:str,
              er_from_factor:str,
              er_rechargeble_battery:bool,
              er_country_origin:str
              ):

```

```
        if type(er_brand)!=str:
            raise TypeError("er_brand must be in str")
        if type(er_cable_feature)!=str:
            raise TypeError("ep_cable_feature must be in str")
        if type(er_net_quantity_in_piece)!=int:
            raise TypeError("er_net_quantity_in_piece must be in str")
        if er_net_quantity_in_piece<=0:
            raise ValueError("er_net_quantity_in_piece must be positive")
        if type(er_colour)!=str:
            raise TypeError("er_colour must be in str")
        if type(er_battery_backup_in_hrs)!=int:
            raise TypeError("er_battery_backup_in_hrs must be in int")
        if er_battery_backup_in_hrs<=0:
            raise TypeError("er_battery_backup_in_hrs must be positive")
        if type(er_control_method)!=str:
            raise TypeError("er_control_method must be str")
        if type(er_nr_of_items)!=int:
            raise TypeError("er_nr_of_items must be int")
        if er_nr_of_items<=0:
            raise ValueError("er_nr_of_items must be positive")
        if type(er_manufacturer)!=str:
            raise TypeError("er_manufacturer must be in str")
        if type(er_model)!=str:
            raise TypeError("er_model must be str")
        if type(er_prod_dimensions)!=ProductDimension:
            raise TypeError("er_prod_dimensions must be in ProductDimension")
        if type(er_batteries)!=str:
            raise TypeError("er_batteries must be in str")
        if type(er_model_nr)!=str:
            raise TypeError("er_model_nr must be in str")
        if type(er_special_feature)!=str:
            raise TypeError("er_special_feature must be in str")
        if '__iter__' not in dir(type(er_mounting_hardware)):
            raise TypeError("er_mounting_hardware must be itarable")
        for hardware in er_mounting_hardware:
            if type(hardware)!=str:
                raise TypeError("hardware must be in str")
        if type(er_batteries_included)!=bool:
            raise TypeError("er_batteries_included must be bool")
        if type(er_batteries_required)!=bool:
            raise TypeError("er_batteries_required must be bool")
        if type(er_batteries_cell_composition)!=str:
            raise TypeError("er_batteries_cell_composition must be str")
        if type(er_from_factor)!=str:
            raise TypeError("er_from_factor must be str")
        if type(er_rechargeble_battery)!=bool:
            raise TypeError("er_rechargeble_battery must be bool")
        if type(er_country_origin)!=str:
            raise TypeError("er_country_origin must be str")

        self.er_brand=er_brand
        self.er_cable_feature=er_cable_feature
        self.er_net_quantity_in_piece=er_net_quantity_in_piece
        self.er_colour=er_colour
        self.er_battery_backup_in_hrs=er_battery_backup_in_hrs
        self.er_control_method=er_control_method
        self.er_nr_of_items=er_nr_of_items
        self.er_manufacturer=er_manufacturer
        self.er_model=er_model
        self.er_prod_dimensions=er_prod_dimensions
        self.er_batteries=er_batteries
        self.er_model_nr=er_model_nr
        self.er_special_feature=er_special_feature
        self.er_mounting_hardware=er_mounting_hardware
        self.er_batteries_included=er_batteries_included
        self.er_batteries_required=er_batteries_required
        self.er_batteries_cell_composition=er_batteries_cell_composition
        self.er_from_factor=er_from_factor
        self.er_rechargeble_battery=er_rechargeble_battery
        self.er_country_origin=er_country_origin
```

```

    #Getter Method

    def get_er_brand(self)->str:
        """
            Returns the er_brand attribute of the calling object
        """
        return self.er_brand
    def get_er_cable_feature(self)->str:
        """
            Returns the er_cable_feature attribute of the calling object
        """
        return self.er_cable_feature

    def get_er_net_quantity_in_piece(self)->int:
        """
            Returns the er_net_quantity_in_piece attribute of the calling object
        """
        return self.er_net_quantity_in_piece

    def get_er_colour(self)->str:
        """
            Returns the er_colour attribute of the calling object
        """
        return self.er_colour

    def get_er_battery_backup_in_hrs(self)->int:
        """
            Returns the er_battery_backup_in_hrs attribute of the calling object
        """
        return self.er_battery_backup_in_hrs

    def get_er_control_method(self)->str:
        """
            Returns the er_control_method attribute of the calling object
        """
        return self.er_control_method

    def get_er_nr_of_items(self)->int:
        """
            Returns the er_nr_of_items attribute of the calling object
        """
        return self.er_nr_of_items

    def get_er_manufacturer(self)->str:
        """
            Returns the er_manufacturer attribute of the calling object
        """
        return self.er_manufacturer

    def get_er_model(self)->str:
        """
            Returns the er_model attribute of the calling object
        """
        return self.er_model

    def get_er_prod_dimensions(self)->ProductDimension:
        """
            Returns the er_prod_dimensions attribute of the calling object
        """
        return self.er_prod_dimensions

    def get_er_batteries(self):
        """
            Returns the er_batteries attribute of the calling object
        """
        return self.er_batteries

    def get_er_model_nr(self)->str:
        """
```

```
            Returns the er_model_nr attribute of the calling object
        """
        return self.er_model_nr

    def get_er_special_feature(self)->str:
        """
            Returns the er_special_feature attribute of the calling object
        """
        return self.er_special_feature

    def get_er_mounting_hardware(self)->[str]:
        """
            Returns the er_special_feature attribute of the calling object
        """
        return self.er_mounting_hardware

    def get_er_batteries_included(self)->bool:
        """
            Returns the er_batteries_included attribute of the calling object
        """
        return self.er_batteries_included

    def get_er_batteries_required(self)->bool:
        """
            Returns the er_batteries_required attribute of the calling object
        """
        return self.er_batteries_required

    def get_er_batteries_cell_composition(self)->str:
        """
            Returns the er_batteries_cell_composition attribute of the calling object
        """
        return self.er_batteries_cell_composition
    def get_er_from_factor(self)->str:
        """
```

```
            Returns the er_from_factor attribute of the calling object
        """
        return self.er_from_factor

    def get_er_rechargeble_battery(self)->bool:
        """
            Returns the er_rechargeble_battery attribute of the calling object
        """
        return self.er_rechargeble_battery

    def get_er_country_origin(self)->str:
        """
            Returns the er_country_origin attribute of the calling object
        """
        return self.er_country_origin

    #Setter Method

    def set_er_brand(self,new_er_brand):
        """
            Sets the er_brand attribute of the calling object to @new_er_cable_feature
            Before setting, TypeCheck is performed and this function returns nothing
        """
        if type(new_er_brand)!=str:
            raise TypeError("new_er_brand must be in str")
        self.er_brand=new_er_brand

    def set_er_cable_feature(self,new_er_cable_feature)->None:
        """
            Sets the er_cable_feature attribute of the calling object to @new_er_cable_feature
            Before setting, TypeCheck is performed and this function returns nothing
        """
        if type(new_er_cable_feature)!=str:
            raise TypeError("new_er_cable_feature must be str")
        self.er_cable_feature=new_er_cable_feature
    def set_er_net_quantity_in_piece(self,new_er_net_quantity_in_piece)->None:
        """
            Sets the er_net_quantity_in_piece attribute of the calling object to
@new_er_net_quantity_in_piece
            Before setting, TypeCheck and ValueCheck is performed.and this
            function returns nothing
        """
        if type(new_er_net_quantity_in_piece)!=int:
            raise TypeError("new_er_net_quantity_in_piece must be in int")
        if new_er_net_quantity_in_piece<=0:
            raise ValueError("new_er_net_quantity_in_piece must be positive")
        self.er_net_quantity_in_piece=new_er_net_quantity_in_piece
    def set_er_colour(self,new_er_colour)->None:
        """
            Sets the er_colour attribute of the calling object to @new_er_colour
            Before setting, TypeCheck is performed and this function returns nothing
        """
        if type(new_er_colour)!=str:
            raise TypeError("new_er_colour must be in str")
        self.er_colour=new_er_colour

    def set_er_battery_backup_in_hrs(self,new_er_battery_backup_in_hrs)->None:
        """
            Sets the er_battery_backup_in_hrs attribute of the calling object to
@new_er_battery_backup_in_hrs
            Before setting, TypeCheck and ValueCheck is performed.and this
            function returns nothing
        """
        if type(new_er_battery_backup_in_hrs)!=int:
            raise TypeError("new_er_battery_backup_in_hrs must be in str")
        if new_er_battery_backup_in_hrs<=0:
            raise ValueError("new_er_battery_backup_in_hrs must be positive")
        self.er_battery_backup_in_hrs=new_er_battery_backup_in_hrs

    def set_er_control_method(self,new_er_control_method)->None:
        """
```

```
            Sets the er_control_method attribute of the calling object to @new_er_control_method
            Before setting, TypeCheck is performed and this function returns nothing
        """
        if type(new_er_control_method)!=str:
            raise TypeError("new_er_control_method must be in str")
        self.er_control_method=new_er_control_method

    def set_er_nr_of_items(self,new_er_nr_of_items)->None:
        """
            Sets the er_nr_of_items attribute of the calling object to @new_er_nr_of_items
            Before setting, TypeCheck and ValueCheck is performed.and this
            function returns nothing
        """
        if type(new_er_nr_of_items)!=int:
            raise TypeError("new_er_nr_of_items must be int")
        if new_er_nr_of_items<=0:
            raise ValueError("er_nr_of_items must be positive")
        self.er_nr_of_items=new_er_nr_of_items

    def set_er_manufacturer(self,new_er_manufacturer)->None:
        """
            Sets the er_manufacturer attribute of the calling object to @new_er_manufacturer
            Before setting, TypeCheck is performed and this function returns nothing
        """
        if(new_er_manufacturer)!=str:
            TypeError("new_er_manufacturer must be in str")
        self.er_manufacturer=new_er_manufacturer

    def set_er_model(self,new_er_model)->None:
        """
            Sets the er_model attribute of the calling object to @new_er_model
            Before setting, TypeCheck is performed and this function returns nothing
        """
        if type(new_er_model)!=str:
            raise TypeError("new_er_model must be in str")
        self.er_model=new_er_model

    def set_er_prod_dimensions(self,new_er_prod_dimensions)->None:
        """
            Sets the er_prod_dimensions attribute of the calling object to @new_er_prod_dimensions
            Before setting, TypeCheck is performed and this function returns nothing
        """
        if type(new_er_prod_dimensions)!=ProductDimension:
            raise TypeError("new_er_prod_dimensions must be in ProductDimension")
        self.er_prod_dimensions=new_er_prod_dimensions

    def set_er_batteries(self,new_er_batteries)->None:
        """
            Sets the er_batteries attribute of the calling object to @new_er_batteries
            Before setting, TypeCheck is performed and this function returns nothing
        """
        if type(new_er_batteries)!=str:
            raise TypeError("new_er_batteries must be in str")
        self.er_batteries=new_er_batteries

    def set_er_model_nr(self,new_er_model_nr)->None:
        """
            Sets the er_model_nr attribute of the calling object to @new_er_model_nr
            Before setting, TypeCheck is performed and this function returns nothing
        """
        if type(new_er_model_nr)!=str:
            raise TypeError("new_er_model_nr must be in str")
        self.er_model_nr=new_er_model_nr

    def set_er_special_feature(self,new_er_special_feature)->None:
        """
            Sets the er_special_feature attribute of the calling object to @new_er_special_feature
            Before setting, TypeCheck is performed and this function returns nothing
        """
        if type(new_er_special_feature)!=str:
            raise TypeError("new_er_special_feature must be in str")
        self.er_special_feature=new_er_special_feature

    def set_er_mounting_hardware(self,new_er_mounting_hardware)->None:
        """
            Sets the er_mounting_hardware attribute of the calling object to @new_er_mounting_hardware
            Before setting, TypeCheck is performed and this function returns nothing
        """

        if '__iter__' not in dir(type(new_er_mounting_hardware)):
            raise TypeError("new_er_mounting_hardware must be iterable")
        for hardware in new_er_mounting_hardware:
            if type(hardware)!=str:
                raise TypeError("hardware must be in str")
        self.er_mounting_hardware=new_er_mounting_hardware
    def set_er_batteries_included(self,new_er_batteries_included)->None:
        """
            Sets the er_batteries_included attribute of the calling object to @new_er_batteries_included
            Before setting, TypeCheck is performed and this function returns nothing
        """
        if type(new_er_batteries_included)!=bool:
            raise TypeError("new_er_batteries_included must be in bool")
        self.er_batteries_included=new_er_batteries_included

    def set_er_batteries_required(self,new_er_batteries_required)->None:
        """
            Sets the er_batteries_required attribute of the calling object to @new_er_batteries_required
            Before setting, TypeCheck is performed and this function returns nothing
        """
        if type(new_er_batteries_required)!=bool:
            raise TypeError("new_er_batteries_required must be in bool")
        self.er_batteries_required=new_er_batteries_required

    def set_er_batteries_cell_composition(self,new_er_batteries_cell_composition)->None:
        """
```

```
            Sets the er_batteries_cell_composition attribute of the calling object to @new_er_batteries_cell_composition
            Before setting, TypeCheck is performed and this function returns nothing
        """
        if type(new_er_batteries_cell_composition)!=str:
            raise TypeError("new_er_batteries_cell_composition must be in str")
        self.er_batteries_cell_composition=new_er_batteries_cell_composition

    def set_er_from_factor(self,new_er_from_factor)->None:
        """
            Sets the er_from_factor attribute of the calling object to @new_er_from_factor
            Before setting, TypeCheck is performed and this function returns nothing
        """
        if type(new_er_from_factor)!=str:
            raise TypeError("new_er_from_factor must be in str")
        self.er_from_factor=new_er_from_factor

    def set_er_rechargeble_battery(self,new_er_rechargeble_battery)->None:
        """
            Sets the er_rechargeble_battery attribute of the calling object to @new_er_rechargeble_battery
            Before setting, TypeCheck is performed and this function returns nothing
        """
        if type(new_er_rechargeble_battery)!=bool:
            raise TypeError("new_er_rechargeble_battery must be in str")
        self.er_rechargeble_battery=new_er_rechargeble_battery

    def set_er_country_origin(self,new_er_country_origin)->None:
        """
            Sets the er_country_origin attribute of the calling object to @new_er_country_origin
            Before setting, TypeCheck is performed and this function returns nothing
        """
        if type(new_er_country_origin)!=str:
            raise TypeError("new_er_country_origin must be in str")
        self.er_country_origin=new_er_country_origin
    def show_earphone_details(self):
        print("Brand:{}".format(self.er_brand))
        print("Cable Feature: {}".format(self.er_cable_feature))
        print("Net Quantity: {}".format(self.er_net_quantity_in_piece))
        print("Colour: {}".format(self.er_colour))
        print("Battery Backup in hours: {}".format(self.er_battery_backup_in_hrs))
        print("Control Method: {}".format(self.er_control_method))
        print("Number of items: {}".format(self.er_nr_of_items))
        print("Manufacturer: {}".format(self.er_manufacturer))
        print("Model: {}".format(self.er_model))
        print("Product Dimensions: {}".format(self.er_prod_dimensions.__dict__))
        print("Batteries: {}".format(self.er_batteries))
        print("Item model number: {}".format(self.er_model_nr))
        print("Special Features: {}".format(self.er_special_feature))
        print("Mounting Hardware:{}".format(self.er_mounting_hardware))
        print("Batteries Included: {}".format(self.er_batteries_included))
        print("Batteries Required: {}".format(self.er_batteries_required))
        print("Batteries Cell Composition:{}".format(self.er_batteries_cell_composition))
        print("From Factor: {}".format(self.er_from_factor))
        print("Includes Rechargeble Battery:{}".format(self.er_rechargeble_battery))
        print("Country of Origin: {}".format(self.er_country_origin))


def main():
    er_obj=Earphone(
            "Boat",
            "Without Cable",
            1,
            "Black",
            60,
            "Voice",
            4,
            "Boat",
            "Rockerz 255 Pro+",
            ProductDimension(45.0,1.0,1.0,45.0),
            "1 Lithium Ion batteries required(included)",
            "Rockerz 255 Pro+",
            "Built-In Voice Assistant",
            "Rockerz 255 Pro +, Charging Cable, Warranty Card, User Manual",
            True,
            True,
            "Lithium Ion",
            "In ear",
            True,
            "India")
    print("EARPHONE PRODUCT DETAILS:")
    er_obj.show_earphone_details()
    #we can also get the attribute using getter method and
    #set the specific attribute using setter method

    sys.exit(0)
main()

"""
Output:-
EARPHONE PRODUCT DETAILS:
Brand:Boat
Cable Feature: Without Cable
Net Quantity: 1
Colour: Black
Battery Backup in hours: 60
Control Method: Voice
Number of items: 4
Manufacturer: Boat
Model: Rockerz 255 Pro+
Product Dimensions: {'length': 45.0, 'width': 1.0, 'height': 1.0, 'weight': 45.0}
Batteries: 1 Lithium Ion batteries required(included)
Item model number: Rockerz 255 Pro+
Special Features: Built-In Voice Assistant
Mounting Hardware:Rockerz 255 Pro +, Charging Cable, Warranty Card, User Manual
Batteries Included: True
Batteries Required: True
Batteries Cell Composition:Lithium Ion
```

```
From Factor: In ear
Includes Rechargeble Battery:True
Country of Origin: India
"""

```