# Class Design Strategy In Python

# Crompton Fan Class

```
----------------------------------------------------------------------------------------------------------------------
"""
   @Author:Kunal Narkhede
   @Date:22/12/2023
   @Goal:To Implement Class Fan
   Capture Real Life Product on Amazon
   http://surl.li/onctm
 """
----------------------------------------------------------------------------------------------------------------------

import sys

class ProductDimension:
   """
      This Class Implement The Dimension Of Crompton Fan
      @__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

    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 Fan:
    def __init__(
            self,
            fan_brand:str,
            fan_colour:str,
            electric_fan_design:str,
            fan_power_source:str,
            fan_style:str,
            fan_prod_dimensions:ProductDimension,
            room_type:[str],
            fan_feature:str,
            fan_mounting_type:str,
            fan_controller_type:str,
            fan_material:str,
            fan_nr_of_speed:int,
            fan_wattage:int,
            fan_finish_type:str,
            fan_nr_of_blades:int,
            fan_air_flow_capacity:str,
            fan_speed:int,
            fan_switch_type:str,
            fan_included_components:[str],
             fan_model_name:str,
             fan_specification_met:str,
             fan_blade_material:str,
             fan_manufacturer:str,
             fan_country_of_origin:str,
             fan_model_num:str,
             fan_ASIN:str
         ):


        if type(fan_brand)!=str:
            raise TypeError("brand must be in str")
        if type(fan_colour)!=str:
            raise TypeError("colour must be in str")
        if type(electric_fan_design)!=str:
            raise TypeError("electric_fan_design must be in str")
        if type(fan_power_source)!=str:
            raise TypeError("power source must be in str")
        if type(fan_style)!=str:
            raise TypeError("style must be in  str")
        if type(fan_prod_dimensions)!=ProductDimension:
            raise TypeError("Product dimension must in Product Dimension")
        if '__iter__' not in dir(type(room_type)):
            raise TypeError("room type must be iterable")
        for room in room_type:
            if type(room)!=str:
                raise TypeError("room must be str")
        if type(fan_feature)!=str:
            raise TypeError("feature must be in str")
        if type(fan_mounting_type)!=str:
            raise TypeError("mounting type must be in str")
        if type(fan_controller_type)!=str:
            raise TypeError("controller type must be in str")
        if type(fan_material)!=str:
            raise TypeError("material must be in str")
        if type(fan_nr_of_speed)!=int:
            raise TypeError("number of speed must be in int")
        if fan_nr_of_speed<=0:
            raise ValueError("number of speed must be positive")
        if type(fan_wattage)!=int:
            raise TypeError("wattage must be in int")
        if fan_wattage<=0:
            raise ValueError("wattage must be in positive")
        if type(fan_finish_type)!=str:
            raise TypeError("finish_type must be in str")
        if type(fan_nr_of_blades)!=int:
            raise TypeError("number of blades must be int")
        if fan_nr_of_blades<=0:
            raise ValueError("number of blades must be positive")
        if type(fan_air_flow_capacity)!=str:
            raise TypeError("air flow capacity must be in str")
        if type(fan_speed)!=int:
            raise TypeError("speed must be in int")
        if fan_speed<=0:
            raise ValueError("speed must be positive")
        if type(fan_switch_type)!=str:
            raise TypeError("switch type must be in str")
        if '__iter__' not in dir(type(fan_included_components)):
            raise TypeError("included components must be iterable")
        for component in fan_included_components:
            if type(component)!=str:
                raise TypeError("component must be str")
        if type(fan_model_name)!=str:
            raise TypeError("model name must be str")
        if type(fan_specification_met)!=str:
            raise TypeError("specification must be str")
        if type(fan_blade_material)!=str:
            raise TypeError("blade materail must be str")
        if type(fan_manufacturer)!=str:
            raise TypeError("manufacturer must be str")
        if type(fan_country_of_origin)!=str:
            raise TypeError("country of origin must be str")
        if type(fan_model_num)!=str:
            raise TypeError("model number must be str")
        if type(fan_ASIN)!=str:
            raise TypeError("ASIN must be str")

        self.fan_brand=fan_brand
        self.fan_colour=fan_colour
        self.electric_fan_design=electric_fan_design
        self.fan_power_source=fan_power_source
        self.fan_style=fan_style
        self.fan_prod_dimensions=fan_prod_dimensions
        self.room_type=room_type
        self.fan_feature=fan_feature
        self.fan_mounting_type=fan_mounting_type
        self.fan_controller_type=fan_controller_type
        self.fan_material=fan_material
        self.fan_nr_of_speed=fan_nr_of_speed
        self.fan_wattage=fan_wattage
        self.fan_finish_type=fan_finish_type
        self.fan_nr_of_blades=fan_nr_of_blades
        self.fan_air_flow_capacity=fan_air_flow_capacity
        self.fan_speed=fan_speed
        self.fan_switch_type=fan_switch_type
        self.fan_included_components=fan_included_components
        self.fan_model_name=fan_model_name
        self.fan_specification_met=fan_specification_met
        self.fan_blade_material=fan_blade_material
        self.fan_manufacturer=fan_manufacturer
        self.fan_country_of_origin=fan_country_of_origin
        self.fan_model_num=fan_model_num
        self.fan_ASIN=fan_ASIN
    #Getter method
    def get_fan_brand(self)->str:
        """
            Returns the fan_brand attribute of the calling object
        """
        return self.fan_brand
    def get_fan_colour(self)->str:
        """
            Returns the fan_colour attribute of the calling object
        """
        return self.fan_colour

    def get_electric_fan_design(self)->str:
        """
            Returns the electric_fan_design attribute of the calling object
        """
        return self.electric_fan_design
    def get_fan_power_source(self)->str:
        """
            Returns the fan_power_source attribute of the calling object
        """
        return self.fan_power_source
    def get_fan_style(self)->str:
        """
            Returns the fan_style attribute of the calling object
        """
        return self.fan_style
    def get_fan_prod_dimensions(self)->ProductDimension:
        """
            Returns the fan_prod_dimensions attribute of the calling object
        """
        return self.fan_prod_dimensions
    def get_room_type(self)->[str]:
        """
            Returns the room_type attribute of the calling object
        """
        return self.room_type
    def get_fan_feature(self)->str:
        """
            Returns the fan_feature attribute of the calling object
        """
        return self.fan_feature
    def get_fan_mounting_type(self)->str:
        """
            Returns the fan_mounting_type attribute of the calling object
        """
        return self.fan_mounting_type
    def get_fan_controller_type(self)->str:
        """
            Returns the fan_controller_type attribute of the calling object
        """
        return self.fan_controller_type
    def get_fan_material(self)->str:
        """
            Returns the fan_material attribute of the calling object
        """
        return self.fan_material
    def get_fan_nr_of_speed(self)->int:
        """
            Returns the fan_nr_of_speed attribute of the calling object
        """
        return self.fan_nr_of_speed
    def get_fan_wattage(self)->int:
        """
            Returns the fan_wattage attribute of the calling object
        """
        return self.fan_wattage
    def get_fan_finish_type(self)->str:
        """
            Returns the fan_finish_type attribute of the calling object
        """
        return self.fan_finish_type
```

```
def get_fan_nr_of_blades(self)->int:
    """
        Returns the fan_nr_of_blades attribute of the calling object
    """
    return self.fan_nr_of_blades
def get_fan_air_flow_capacity(self)->str:
    """
        Returns the fan_air_flow_capacity attribute of the calling object
    """
    return self.fan_air_flow_capacity
def get_fan_speed(self)->int:
    """
        Returns the fan_speed attribute of the calling object
    """
    return self.fan_speed
def get_fan_switch_type(self)->str:
    """
        Returns the fan_switch_type attribute of the calling object
    """
    return self.fan_switch_type
def get_fan_included_components(self)->[str]:
    """
        Returns the fan_included_components attribute of the calling object
    """
    return self.fan_included_components
def get_fan_model_name(self)->str:
    """
        Returns the fan_model_name attribute of the calling object
    """
    return self.fan_model_name
def get_fan_specification_met(self)->str:
    """
        Returns the fan_specification_met attribute of the calling object
    """
    return self.fan_specification_met
def get_fan_blade_material(self)->str:
    """
        Returns the fan_blade_material attribute of the calling object
    """
    return self.fan_blade_material
def get_fan_manufacturer(self)->str:
    """
        Returns the fan_manufacturer attribute of the calling object
    """
    return self.fan_manufacturer
def get_fan_country_of_origin(self)->str:
    """
        Returns the fan_country_of_origin attribute of the calling object
    """
    return self.fan_country_of_origin
def get_fan_model_num(self)->str:
    """
        Returns the fan_model_num attribute of the calling object
    """
    return self.fan_model_num
def get_fan_ASIN(self)->str:
    """
        Returns the fan_ASIN attribute of the calling object
    """
    return self.fan_ASIN
#Setter method
def set_fan_brand(self,new_fan_brand)->None:
    """
        Sets the fan_brand attribute of the calling object to @new_fan_brand
        Before setting, TypeCheck is performed.
    """
    if type(new_fan_brand)!=str:
        raise TypeError("new_fan_brand must be str")
    self.fan_brand=new_fan_brand

def set_fan_colour(self,new_fan_colour)->None:
    """
        Sets the fan_colour attribute of the calling object to @new_fan_colour
        Before setting, TypeCheck is performed.
    """
    if type(new_fan_colour)!=str:
        raise TypeError("new_fan_colour must be str")
    self.fan_colour=new_fan_colour

def set_electric_fan_design(self,new_electric_fan_design)->None:
    """
        Sets the electric_fan_design attribute of the calling object to @new_electric_fan_design
        Before setting, TypeCheck is performed.
    """
    if type(new_electric_fan_design)!=str:
        raise TypeError("new_electric_fan_design must be str")
    self.electric_fan_design=new_electric_fan_design

def set_fan_power_source(self,new_fan_power_source)->None:
    """
        Sets the fan_power_source attribute of the calling object to @new_fan_power_source
        Before setting, TypeCheck is performed.
    """
    if type(new_fan_power_source)!=str:
        raise TypeError("new_fan_power_source must be str")
    self.fan_power_source=new_fan_power_source

def set_fan_style(self,new_fan_style)->None:
    """
        Sets the fan_style attribute of the calling object to @new_fan_style
        Before setting, TypeCheck is performed.
    """
    if type(new_fan_style)!=str:
        raise TypeError("new_fan_style must be str")
    self.fan_style=new_fan_style
def set_fan_prod_dimensions(self,new_fan_prod_dimensions)->None:
    """
        Sets the fan_prod_dimensions attribute of the calling object to @new_fan_prod_dimensions
        Before setting, TypeCheck is performed.
    """
    if type(new_fan_prod_dimensions)!=str:
        raise TypeError("new_fan_prod_Dimensions must be ProductDimension")
    self.fan_prod_dimensions=new_fan_prod_dimensions

def set_room_type(self,new_room_type)->None:
    """
        Sets the room_type attribute of the calling object to @new_room_type
        Before setting, TypeCheck is performed.
    """
    if '__iter__' not in dir(type(new_room_type)):
        raise TypeError("new_room_type must be iterable")
    for room in new_room_type:
        if type(room)!=str:
            raise TypeError("room must be str")
    self.room_type=new_room_type

def set_fan_feature(self,new_fan_feature)->None:
    """
        Sets the fan_feature attribute of the calling object to @new_fan_feature
        Before setting, TypeCheck is performed.
    """
    if type(new_fan_feature)!=str:
        raise TypeError("new_fan_feature must be str")
    self.fan_feature=new_fan_feature

def set_fan_mounting_type(self,new_mounting_type)->None:
    """
        Sets the mounting_type attribute of the calling object to @new_mounting_type
        Before setting, TypeCheck is performed.
    """
    if type(new_mounting_type)!=str:
        raise TypeError("new_mounting_type must be str")
    self.fan_mounting_type=new_mounting_type

def set_fan_controller_type(self,new_fan_controller_type)->None:
    """
        Sets the fan_controller_type attribute of the calling object to @new_fan_controller_type
        Before setting, TypeCheck is performed.
    """
    if type(new_fan_controller_type)!=str:
        raise TypeError("new_fan_controller_type must be str")
    self.fan_controller_type=new_fan_controller_type

def set_fan_material(self,new_fan_material)->None:
    """
        Sets the fan_material attribute of the calling object to @new_fan_material
        Before setting, TypeCheck is performed.
    """
    if type(new_fan_material)!=str:
        raise TypeError("new_fan_material must be str")
    self.fan_material=new_fan_material

def set_fan_nr_of_speed(self,new_fan_nr_of_speed)->None:
    """
        Sets the fan_nr_of_speed attribute of the calling object to @new_fan_nr_of_speed
        Before setting, TypeCheck and ValueCheck is performed.
    """
    if type(new_fan_nr_of_speed)!=int:
        raise TypeError("new_fan_nr_of_speed must be int")
    if new_fan_nr_of_speed<=0:
        raise ValueError("new_fan_nr_of_speed must be positive")
    self.fan_nr_of_speed=new_fan_nr_of_speed

def set_fan_wattage(self,new_fan_wattage)->None:
    """
        Sets the fan_wattage attribute of the calling object to @new_fan_wattage
        Before setting, TypeCheck and ValueCheck is performed.
    """
    if type(new_fan_wattage)!=int:
        raise TypeError("new_fan_wattage must be int")
    if new_fan_wattage<=0:
        raise ValueError("new_fan_wattage must be positive")
    self.fan_wattage=new_fan_wattage

def set_fan_finish_type(self,new_fan_finish_type)->None:
    """
        Sets the fan_finish_type attribute of the calling object to @new_fan_finish_type
        Before setting, TypeCheck is performed.
    """
    if type(new_fan_finish_type)!=str:
        raise TypeError("new_fan_finish_type must be str")
    self.fan_finish_type=new_fan_finish_type

def set_fan_nr_of_blades(self,new_fan_nr_of_blades):
    """
        Sets the fan_nr_of_blades attribute of the calling object to @new_fan_nr_of_blades
        Before setting, TypeCheck and ValueCheck is performed.
    """
    if type(new_fan_nr_of_blades)!=int:
        raise TypeError("new_fan_nr_of_blades must be int")
    if new_fan_nr_of_blades<=0:
        raise ValueError("new_fan_nr_of_blades must be positive")
    self.fan_nr_of_blades=new_fan_nr_of_blades

def set_fan_air_flow_capacity(self,new_air_flow_capacity)->None:
    """
        Sets the air_flow_capacity attribute of the calling object to @new_air_flow_capacity
        Before setting, TypeCheck is performed.
    """
    if type(new_air_flow_capacity)!=str:
        raise TypeError("new_air_flow_capacity must be str")
    self.fan_air_flow_capacity=new_air_flow_capacity

def set_fan_speed(self,new_fan_speed)->None:
    """
        Sets the fan_speed attribute of the calling object to @new_fan_speed
        Before setting, TypeCheck and ValueCheck is performed.
    """
    if type(new_fan_speed)!=int:
        raise TypeError("new_fan_speed must be int")
    if new_fan_speed<=0:
        raise ValueError("new_fan_speed must be positive")
    self.fan_speed=new_fan_speed

def set_fan_switch_type(self,new_fan_switch_type)->None:
    """
        Sets the fan_switch_type attribute of the calling object to @new_fan_switch_type
        Before setting, TypeCheck is performed.
    """
    if type(new_fan_switch_type)!=str:
        raise TypeError("new_fan_switch_type must be str")
    self.fan_switch_type=new_fan_switch_type

def set_fan_included_components(self,new_fan_included_components)->None:
    """
        Sets the fan_included_components attribute of the calling object to
@new_fan_included_components
        Before setting, TypeCheck is performed.
    """
    if '__iter__' not in dir(type(new_fan_included_components)):
        raise TypeError("new_fan_included_components must itetable")
    for component in new_fan_included_components:
        if type(component)!=str:
            raise TypeError("feature must be str")
    self.fan_included_components=new_fan_included_components


def set_fan_model_name(self,new_fan_model_name)->None:
    """
        Sets the fan_model_name attribute of the calling object to @new_fan_model_name
        Before setting, TypeCheck is performed.
    """
    if type(new_fan_model_name)!=str:
        raise TypeError("new_fan_model_name must be str")
    self.fan_model_name=new_fan_model_name

def set_fan_specification_met(self,new_fan_specification_met)->None:
    """
        Sets the fan_specification_met attribute of the calling object to @new_fan_specification_met
        Before setting, TypeCheck is performed.
    """
    if type(new_fan_specification_met)!=str:
        raise TypeError("new_fan_specification_met must be str")
    self.fan_specification_met=new_fan_specification_met

def set_fan_blade_material(self,new_fan_blade_material)->None:
    """
        Sets the fan_blade_material attribute of the calling object to @new_fan_blade_material
        Before setting, TypeCheck is performed.
    """
    if type(new_fan_blade_material)!=str:
        raise TypeError("new_fan_blade_material must be str")
    self.fan_blade_material=new_fan_blade_material

def set_fan_manufacturer(self,new_fan_manufacturer):
    """
        Sets the fan_manufacturer attribute of the calling object to @new_fan_manufacturer
        Before setting, TypeCheck is performed.
    """
    if type(new_fan_manufacturer)!=str:
        raise TypeError("new_fan_manufacturer must be str")
    self.fan_manufacturer=new_fan_manufacturer

def set_fan_country_of_origin(self,new_fan_country_of_origin)->None:
    """
        Sets the fan_country_of_origin attribute of the calling object to @new_fan_country_of_origin
        Before setting, TypeCheck is performed.
    """
    if type(new_fan_country_of_origin)!=str:
        raise TypeError("new_fan_country_of_origin must be str")
    self.fan_country_of_origin=new_fan_country_of_origin

def set_fan_model_num(self,new_model_num)->None:
    """
        Sets the model_num attribute of the calling object to @new_model_num
        Before setting, TypeCheck is performed.
    """
    if type(new_model_num)!=str:
        raise TypeError("new_model_num must be str")
    self.fan_model_num=new_model_num

def set_fan_ASIN(self,new_fan_ASIN)->None:
    """
        Sets the fan_ASIN attribute of the calling object to @new_fan_ASIN
        Before setting, TypeCheck is performed.
    """
    if type(new_fan_ASIN)!=str:
        raise TypeError("new_fan_ASIN must be str")
    self.fan_ASIN=new_fan_ASIN
def show(self)->None:

    """
        This function display all the characterstics of Fan class
    """
    print("Brand:{}".format(self.fan_brand))
    print("Colour:{}".format(self.fan_colour))
    print("Electric fan design:{}".format(self.electric_fan_design))
    print("Power Source:{}".format(self.fan_power_source))
    print("Style:{}".format(self.fan_style))
    print("Product Dimension:{}".format(self.fan_prod_dimensions.__dict__))
    print("Room type:{}".format(self.room_type))
    print("Special Feature:{}".format(self.fan_feature))
    print("Mounting type:{}".format(self.fan_mounting_type))
    print("Controller type:{}".format(self.fan_controller_type))
    print("Material:{}".format(self.fan_material))

    print("Number of Speeds:{}".format(self.fan_nr_of_speed))
    print("Wattage:{}".format(self.fan_wattage))
    print("Finish Type:{}".format(self.fan_finish_type))
    print("Number of blades:{}".format(self.fan_nr_of_blades))
    print("Air flow capacity:{}".format(self.fan_air_flow_capacity))
    print("Speed :{}".format(self.fan_speed))
    print("Switch type :{}".format(self.fan_switch_type))
    print("Included Components:{}".format(self.fan_included_components))
    print("Model Name:{}".format(self.fan_model_name))

    print("Specification Met:{}".format(self.fan_specification_met))
    print("Blade Material:{}".format(self.fan_blade_material))
    print("Country of Origin:{}".format(self.fan_country_of_origin))
    print("Model number:{}".format(self.fan_model_num))
    print("ASIN number:{}".format(self.fan_ASIN))



def main():
fan_obj=Fan(
        "Crompton",
        "Lustre Brown",
        "Ceiling Fan",
        "Electricity",
        "Sea Sapphira(1 Star Rated)",
        ProductDimension(54.5,25.5,19.4,3.4),
        ["Living Room","Bedroom","Daning Room"],
        "High Velocity",
        "downrod mount",
        "Regulator Control",
        "Aluminium",
        3,
        51,
        "Power Coated",
        3,
        "210 Cubic Matres Minute",
        380,
        "Push Button",
        ["Fan motor","Balanced blade set","downrod","canopies","safety cable and shackle kit
assembly"],
        "CROMPTON SUREBREEZE SEA",
        "CE",
        "CRCA",
        "Crompton Greaves Consumer Electricals Limited",
        "India",
        "CFSBSSP48LB1S",
        "B0BTS9GG2V"
)
print("FAN PRODUCT DETAILS:")
fan_obj.show()
#we can also get the attribute using getter method and
#set the specific attribute using setter method
sys.exit(0)
main()

"""
Ouput:
----------------------------------------------
FAN PRODUCT DETAILS:
Brand:Crompton
```

```
Colour:Lustre Brown
Electric fan design:Ceiling Fan
Power Source:Electricity
Style:Sea Sapphira(1 Star Rated)
Product Dimension:{'length': 54.5, 'width': 25.5, 'height': 19.4, 'weight': 3.4}
Room type:['Living Room', 'Bedroom', 'Daning Room']
Special Feature:High Velocity
Mounting type:downrod mount
Controller type:Regulator Control
Material:Aluminium
Number of Speeds:3
Wattage:51
Finish Type:Power Coated
Number of blades:3
Air flow capacity:210 Cubic Matres Minute
Speed :380
Switch type :Push Button
Included Components:['Fan motor', 'Balanced blade set', 'downrod', 'canopies', 'safety cable and shackle kit assembly']
Model Name:CROMPTON SUREBREEZE SEA
Specification Met:CE
Blade Material:CRCA
Country of Origin:India
Model number:CFSBSSP48LB1S
ASIN number:B0BTS9GG2V
"""
```