In [1]:
```
class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height
        self.area = width*height
        self.peremeter = 2*width + 2*height
```

In [2]:
```
a = Rectangle(5, 7)
a
```

Out[2]:
```
<__main__.Rectangle at 0x2156456fe48>
```

In [3]:
```
print(a.height)
print(a.area)
print(a.peremeter)
```

```
7
35
24
```

In [4]:
```
b = Rectangle(2, 5)
b.area
```

Out[4]:
```
10
```

The classic Shipping Container Example

In [5]:
```
class ShippingContainer:
    def __init__(self, owner_code, contents):
        # owner_code will be a string and contents would be a list of strin
gs
        self.owner_code = owner_code
        self.contents = contents

c1 = ShippingContainer('YML', ['Books'])
print('c1 Owner:', c1.owner_code)
print('c1 Contents:', c1.contents)
```

```
c1 Owner: YML
c1 Contents: ['Books']
```

Class Attributes

These attributes belongs to class rather than a single instance of class.

In [6]:
```
class ShippingContainer:
    next_serial = 1337
    def __init__(self, owner_code, contents):
        # owner_code will be a string and contents would be a list of strin
gs
        self.owner_code = owner_code
        self.contents = contents
        self.serial_number = ShippingContainer.next_serial
        ShippingContainer.next_serial += 1

c1 = ShippingContainer('YML', ['Books'])
print('c1 Serial Number:', c1.serial_number)
print('c1 Owner:', c1.owner_code)
print('c1 Contents:', c1.contents)
print('')
c2 = ShippingContainer('ESC', ['Electronics'])
print('c2 Serial Number:', c2.serial_number)
print('c2 Owner:', c2.owner_code)
print('c2 Contents:', c2.contents)
```

```
c1 Serial Number: 1337
c1 Owner: YML
c1 Contents: ['Books']

c2 Serial Number: 1338
c2 Owner: ESC
c2 Contents: ['Electronics']
```

In [7]:
```
# Interesting: we can get this class attribute using the class name OR the
 instance name
print(ShippingContainer.next_serial)
print(c1.next_serial)
print(c2.next_serial)
```

```
1339
1339
1339
```

- We can use the class attribute with self identifier, i.e self.serial_number = self.next_serial will work just fine.
- But problem occurs when we try to assign value to class attribute using self identifier.
- self.next_serial += 1 will contain a new instance attribute rather than modifying the class attribute.

If class attribute and instance attribute exists with same name, self.name will always give presidence to instance attribute

Static Methods

In [8]:
```
class ShippingContainer:
    next_serial = 1337

    def _generate_serial(self):
        # function name starting with _ as we will never use it outside the
class defenintion
        result = ShippingContainer.next_serial
        ShippingContainer.next_serial += 1
        return result

    def __init__(self, owner_code, contents):
        # owner_code will be a string and contents would be a list of strin
gs
        self.owner_code = owner_code
        self.contents = contents
        self.serial_number = self._generate_serial()

c1 = ShippingContainer('YML', ['Books'])
print('c1 Serial Number:', c1.serial_number)
print('c1 Owner:', c1.owner_code)
print('c1 Contents:', c1.contents)
print('')
c2 = ShippingContainer('ESC', ['Electronics'])
print('c2 Serial Number:', c2.serial_number)
print('c2 Owner:', c2.owner_code)
print('c2 Contents:', c2.contents)
```

```
c1 Serial Number: 1337
c1 Owner: YML
c1 Contents: ['Books']

c2 Serial Number: 1338
c2 Owner: ESC
c2 Contents: ['Electronics']
```

In [9]:
```
# As we can see that the self parameter of the function _generate_serial is
redundant as we never use it.
# We can use static functions when functions are static for each and every
 instance.
class ShippingContainer:
    next_serial = 1337

    @staticmethod
    # self argument removed
    def _generate_serial():
        # function name starting with _ as we will never use it outside the
class definintion
        result = ShippingContainer.next_serial
        ShippingContainer.next_serial += 1
        return result

    def __init__(self, owner_code, contents):
        # owner_code will be a string and contents would be a list of strin
gs
        self.owner_code = owner_code
        self.contents = contents
        # Its considered a good practice to use statics methods with class
 identifiers rather than self
        self.serial_number = ShippingContainer._generate_serial()

c1 = ShippingContainer('YML', ['Books'])
print('c1 Serial Number:', c1.serial_number)
print('c1 Owner:', c1.owner_code)
print('c1 Contents:', c1.contents)
print('')
c2 = ShippingContainer('ESC', ['Electronics'])
print('c2 Serial Number:', c2.serial_number)
print('c2 Owner:', c2.owner_code)
print('c2 Contents:', c2.contents)
```

```
c1 Serial Number: 1337
c1 Owner: YML
c1 Contents: ['Books']

c2 Serial Number: 1338
c2 Owner: ESC
c2 Contents: ['Electronics']
```

In [10]:
```
c1._generate_serial()
c3 = ShippingContainer('AFL', ['Toys'])
c3.serial_number
```

Out[10]:
```
1340
```

In [11]:
```
ShippingContainer._generate_serial()
c4 = ShippingContainer('BGT', ['Meds'])
c4.serial_number
```

Out[11]:
```
1342
```

Class Methods

In [12]:
```python
class ShippingContainer:
    next_serial = 1337
    
    @classmethod
    # class methods can take in class attributes with cls identifiers
    def _generate_serial(cls):
        # It takes cls as an argument
        result = cls.next_serial
        cls.next_serial += 1
        return result
    
    def __init__(self, owner_code, contents):
        # owner_code will be a string and contents would be a list of strin
gs
        self.owner_code = owner_code
        self.contents = contents
        # Its considered a good practice to use statics methods with class
 identifiers rather than self
        self.serial_number = ShippingContainer._generate_serial()

c1 = ShippingContainer('YML', ['Books'])
print('c1 Serial Number:', c1.serial_number)
print('c1 Owner:', c1.owner_code)
print('c1 Contents:', c1.contents)
print('')
c2 = ShippingContainer('ESC', ['Electronics'])
print('c2 Serial Number:', c2.serial_number)
print('c2 Owner:', c2.owner_code)
print('c2 Contents:', c2.contents)
```

```
c1 Serial Number: 1337
c1 Owner: YML
c1 Contents: ['Books']

c2 Serial Number: 1338
c2 Owner: ESC
c2 Contents: ['Electronics']
```

- Use @classmethod when you require access to class attributes and methods.
- Use @staticmethod when you require access to instance attribute and methods.

Factory Method

- Returns instance of class with different combination of arguments.
- These methods allows callers to express intents.

In [13]:
```
class ShippingContainer:
    next_serial = 1337

    @classmethod
    def _generate_serial(cls):
        result = cls.next_serial
        cls.next_serial += 1
        return result

    # Mentioned below are the factory methods to create custom instances wi
thout modification in __init__
    @classmethod
    def create_empty(cls, owner_code):
        return cls(owner_code, contents=[])

    @classmethod
    def create_with_items(cls, owner_code, items):
        return cls(owner_code, contents=list(items))


    def __init__(self, owner_code, contents):
        # owner_code will be a string and contents would be a list of strin
gs
        self.owner_code = owner_code
        self.contents = contents
        self.serial_number = ShippingContainer._generate_serial()

c1 = ShippingContainer('YML', ['Books'])
print('c1 Serial Number:', c1.serial_number)
print('c1 Owner:', c1.owner_code)
print('c1 Contents:', c1.contents)
print('')
c2 = ShippingContainer('ESC', ['Electronics'])
print('c2 Serial Number:', c2.serial_number)
print('c2 Owner:', c2.owner_code)
print('c2 Contents:', c2.contents)
print('')
# Creating a special custom instance using factory method which returns an
 instance
c3 = ShippingContainer.create_empty('BFG')
print('c3 Serial Number:', c3.serial_number)
print('c3 Owner:', c3.owner_code)
print('c3 Contents:', c3.contents)
print('')
c4 = ShippingContainer.create_with_items('ASF', {'Food', 'Eggs', 'Milk'})
print('c4 Serial Number:', c4.serial_number)
print('c4 Owner:', c4.owner_code)
print('c4 Contents:', c4.contents)
```

```
c1 Serial Number: 1337
c1 Owner: YML
c1 Contents: ['Books']

c2 Serial Number: 1338
c2 Owner: ESC
c2 Contents: ['Electronics']

c3 Serial Number: 1339
c3 Owner: BFG
c3 Contents: []

c4 Serial Number: 1340
c4 Owner: ASF
c4 Contents: ['Milk', 'Food', 'Eggs']
```

In [14]:
```python
import iso6346

class ShippingContainer:

    next_serial = 1337

    @classmethod
    def _generate_serial(cls):
        result = cls.next_serial
        cls.next_serial += 1
        return result

    @staticmethod
    def _make_bic_code(owner_code, serial):
        return iso6346.create(
            owner_code=owner_code,
            serial=str(serial).zfill(6)
        )

    @classmethod
    def create_empty(cls, owner_code):
        return cls(owner_code, contents=[])

    @classmethod
    def create_with_items(cls, owner_code, items):
        return cls(owner_code, contents=list(items))

    def __init__(self, owner_code, contents):
        self.owner_code = owner_code
        self.contents = contents
        self.bic = ShippingContainer._make_bic_code(
            owner_code=owner_code,
            serial=ShippingContainer._generate_serial()
        )

c1 = ShippingContainer('YML', ['Books'])
print('c1 BIC Number:', c1.bic)
print('c1 Owner:', c1.owner_code)
print('c1 Contents:', c1.contents)
print('')
c2 = ShippingContainer('ESC', ['Electronics'])
print('c2 BIC Number:', c2.bic)
print('c2 Owner:', c2.owner_code)
print('c2 Contents:', c2.contents)
print('')
# Creating a special custom instance using factory method which returns an
 instance
c3 = ShippingContainer.create_empty('BFG')
print('c3 BIC Number:', c3.bic)
print('c3 Owner:', c3.owner_code)
print('c3 Contents:', c3.contents)
print('')
c4 = ShippingContainer.create_with_items('ASF', {'Food', 'Eggs', 'Milk'})
print('c4 BIC Number:', c4.bic)
print('c4 Owner:', c4.owner_code)
print('c4 Contents:', c4.contents)
```

```
c1 BIC Number: YMLU0013374
c1 Owner: YML
c1 Contents: ['Books']

c2 BIC Number: ESCU0013388
c2 Owner: ESC
c2 Contents: ['Electronics']

c3 BIC Number: BFGU0013390
c3 Owner: BFG
c3 Contents: []

c4 BIC Number: ASFU0013403
c4 Owner: ASF
c4 Contents: ['Milk', 'Food', 'Eggs']
```

Static Method with inheritance

- The 4th letter in bic code specify unclassified.
- Lets introduce a inherited container, this will take all properties of container but will have 4th letter as 'R'
- For the purpose we will use category argument of iso6346 create function.

In [15]:
```python
class ShippingContainer:

    next_serial = 1337

    @classmethod
    def _generate_serial(cls):
        result = cls.next_serial
        cls.next_serial += 1
        return result

    @staticmethod
    def _make_bic_code(owner_code, serial):
        return iso6346.create(
            owner_code=owner_code,
            serial=str(serial).zfill(6)
        )

    @classmethod
    def create_empty(cls, owner_code):
        return cls(owner_code, contents=[])

    @classmethod
    def create_with_items(cls, owner_code, items):
        return cls(owner_code, contents=list(items))

    def __init__(self, owner_code, contents):
        self.owner_code = owner_code
        self.contents = contents
        self.bic = ShippingContainer._make_bic_code(
            owner_code=owner_code,
            serial=ShippingContainer._generate_serial()
        )

# Inheritting ShippingContainer
class RefrigeratedShippingContainer(ShippingContainer):
    # Method overriding
    @staticmethod
    def _make_bic_code(owner_code, serial):
        return iso6346.create(
            owner_code=owner_code,
            serial=str(serial).zfill(6),
            category='R'
        )

r1 = RefrigeratedShippingContainer('YML', ['Fish'])
print('r1 BIC Number:', r1.bic)
print('r1 Owner:', r1.owner_code)
print('r1 Contents:', r1.contents)
```

```
r1 BIC Number: YMLU0013374
r1 Owner: YML
r1 Contents: ['Fish']
```

Why It didn't worked??
Because init method specifically specified that we use mekebiccode method of ShippingContainer Class and not the overridden method.
For polymorphic dispatch invoke static method through self

In [16]:
```python
class ShippingContainer:
    next_serial = 1337
    @classmethod
    def _generate_serial(cls):
        result = cls.next_serial
        cls.next_serial += 1
        return result
    @staticmethod
    def _make_bic_code(owner_code, serial):
        return iso6346.create(
            owner_code=owner_code,
            serial=str(serial).zfill(6)
        )
    @classmethod
    def create_empty(cls, owner_code):
        return cls(owner_code, contents=[])
    @classmethod
    def create_with_items(cls, owner_code, items):
        return cls(owner_code, contents=list(items))
    def __init__(self, owner_code, contents):
        self.owner_code = owner_code
        self.contents = contents
        # Using self below so that overridden function can be used
        # I am still using ShippingContainer._generate_serial because I wan
t next_serial to get updated even if I initialize a refrigerated shipping c
ontainer.
        self.bic = self._make_bic_code(
            owner_code=owner_code,
            serial=ShippingContainer._generate_serial()
        )

# Inheritting ShippingContainer
class RefrigeratedShippingContainer(ShippingContainer):
    # Method overriding
    @staticmethod
    def _make_bic_code(owner_code, serial):
        return iso6346.create(
            owner_code=owner_code,
            serial=str(serial).zfill(6),
            category='R'
        )

r1 = RefrigeratedShippingContainer('YML', ['Fish'])
print('r1 BIC Number:', r1.bic)
print('r1 Owner:', r1.owner_code)
print('r1 Contents:', r1.contents)
```

```
r1 BIC Number: YMLR0013372
r1 Owner: YML
r1 Contents: ['Fish']
```

Few pointers about functions argument:

- ***args are used for positional arguments, these will give out list of undefined positional arguments.**
- **** kwargs are used for undefined key word arguments, these will be given as dict.**
- **def func(a, *, b) here *, means that beyond this point only keyword arguments can be defined.**

Class Method with inheritance

These methods works just fine with sub classes, calling create_empty will create empty instance of Ref. shipping container because cls is passed as argument

In [17]:
```python
c1 = ShippingContainer.create_empty('YML')
print('c1 BIC Number:', c1.bic)
print('c1 Owner:', c1.owner_code)
print('c1 Contents:', c1.contents)
print('')
r2 = RefrigeratedShippingContainer.create_empty('ESC')
print('r2 BIC Number:', r2.bic)
print('r2 Owner:', r2.owner_code)
print('r2 Contents:', r2.contents)
r2
```

```
c1 BIC Number: YMLU0013380
c1 Owner: YML
c1 Contents: []

r2 BIC Number: ESCR0013391
r2 Owner: ESC
r2 Contents: []
```

Out[17]:
```
<__main__.RefrigeratedShippingContainer at 0x215645e5708>
```

How to define attributes and constants for sub-classes

In [18]:
```
# If we want to use extra arguments in our derived classes, we will have to
use use **kwargs in factory methods so it doesn't throws error
class ShippingContainer:
    next_serial = 1337
    @classmethod
    def _generate_serial(cls):
        result = cls.next_serial
        cls.next_serial += 1
        return result
    @staticmethod
    def _make_bic_code(owner_code, serial):
        return iso6346.create(
            owner_code=owner_code,
            serial=str(serial).zfill(6)
        )
    # We are not using undefined keyword arguments in the base class, its j
ust there to prevent error when we use factory method with derived classes.
    @classmethod
    def create_empty(cls, owner_code, **kwargs):
        return cls(owner_code, contents=[], **kwargs)
    @classmethod
    def create_with_items(cls, owner_code, items, **kwargs):
        return cls(owner_code, contents=list(items), **kwargs)
    def __init__(self, owner_code, contents, **kwargs):
        self.owner_code = owner_code
        self.contents = contents
        self.bic = self._make_bic_code(
            owner_code=owner_code,
            serial=ShippingContainer._generate_serial()
        )


class RefrigeratedShippingContainer(ShippingContainer):

    MAX_CELSIUS = 4

    @staticmethod
    def _make_bic_code(owner_code, serial):
        return iso6346.create(
            owner_code=owner_code,
            serial=str(serial).zfill(6),
            category='R'
        )

    # We will first call init of base class with super().init, then we can
 add the extra attributes to the instance
    def __init__(self, owner_code, contents, *, celsius, **kwargs):
        super().__init__(owner_code, contents, **kwargs)
        if celsius > RefrigeratedShippingContainer.MAX_CELSIUS:
            raise ValueError('Temprature is too hot!')
        self.celsius = celsius

r1 = RefrigeratedShippingContainer('YML', ['Fish'], celsius=3.2)
print('r1 BIC Number:', r1.bic)
print('r1 Owner:', r1.owner_code)
print('r1 Contents:', r1.contents)
print('r1 Temprature:', r1.celsius)
```

```
r1 BIC Number: YMLR0013372
r1 Owner: YML
r1 Contents: ['Fish']
r1 Temprature: 3.2
```

In [19]:

```
# Vissible issue
# While we are checking the temprature while initializing, it can still be
 modified later
r1.celsius = 12
print('r1 Temprature:', r1.celsius)
```

```
r1 Temprature: 12
```

How to make handle this problem?

Properties

In [20]:
```
# A defined property acts as an attribute to a class which can be used to g
et an attribute which is not supposed for public use. (getter)
# Defining property without defining its setter makes a read-only attribut
e.
class ShippingContainer:
    next_serial = 1337
    @classmethod
    def _generate_serial(cls):
        result = cls.next_serial
        cls.next_serial += 1
        return result
    @staticmethod
    def _make_bic_code(owner_code, serial):
        return iso6346.create(
            owner_code=owner_code,
            serial=str(serial).zfill(6)
        )
    # We are not using undefined keyword arguments in the base class, its j
ust there to prevent error when we use factory method with derived classes.
    @classmethod
    def create_empty(cls, owner_code, **kwargs):
        return cls(owner_code, contents=[], **kwargs)
    @classmethod
    def create_with_items(cls, owner_code, items, **kwargs):
        return cls(owner_code, contents=list(items), **kwargs)
    def __init__(self, owner_code, contents, **kwargs):
        self.owner_code = owner_code
        self.contents = contents
        self.bic = self._make_bic_code(
            owner_code=owner_code,
            serial=ShippingContainer._generate_serial()
        )


class RefrigeratedShippingContainer(ShippingContainer):
    MAX_CELSIUS = 4

    @staticmethod
    def _make_bic_code(owner_code, serial):
        return iso6346.create(
            owner_code=owner_code,
            serial=str(serial).zfill(6),
            category='R'
        )

    # We are using a property to return a attribute which is not meant for
 public access.
    @property
    def celsius(self):
        return self._celsius
    def __init__(self, owner_code, contents, *, celsius, **kwargs):
        super().__init__(owner_code, contents, **kwargs)
        if celsius > RefrigeratedShippingContainer.MAX_CELSIUS:
            raise ValueError('Temprature is too hot!')
        # Using _celsius to indicate that this attribute should not be used
for public access.
        self._celsius = celsius

r1 = RefrigeratedShippingContainer('YML', ['Fish'], celsius=3.2)
print('r1 Temprature:', r1.celsius)

# However if we try to assign value to r1.celsius we will get a 'cant set a
ttribute error.'
```

```
r1 Temprature: 3.2
```

In [21]:
```
# Defining property setter
class ShippingContainer:
    next_serial = 1337
    @classmethod
    def _generate_serial(cls):
        result = cls.next_serial
        cls.next_serial += 1
        return result
    @staticmethod
    def _make_bic_code(owner_code, serial):
        return iso6346.create(
            owner_code=owner_code,
            serial=str(serial).zfill(6)
        )
    # We are not using undefined keyword arguments in the base class, its j
ust there to prevent error when we use factory method with derived classes.
    @classmethod
    def create_empty(cls, owner_code, **kwargs):
        return cls(owner_code, contents=[], **kwargs)
    @classmethod
    def create_with_items(cls, owner_code, items, **kwargs):
        return cls(owner_code, contents=list(items), **kwargs)
    def __init__(self, owner_code, contents, **kwargs):
        self.owner_code = owner_code
        self.contents = contents
        self.bic = self._make_bic_code(
            owner_code=owner_code,
            serial=ShippingContainer._generate_serial()
        )


class RefrigeratedShippingContainer(ShippingContainer):
    MAX_CELSIUS = 4

    @staticmethod
    def _make_bic_code(owner_code, serial):
        return iso6346.create(
            owner_code=owner_code,
            serial=str(serial).zfill(6),
            category='R'
        )

    # We are using a property to return a attribute which is not meant for
 public access.
    @property
    def celsius(self):
        return self._celsius

    # We can define a setter for a property after defining a property.
    # Using this we will we able to create a writable property.
    # AND HERE we can define the property update conditions....which was ou
r main issue
    @celsius.setter
    def celsius(self, value):
        if value > RefrigeratedShippingContainer.MAX_CELSIUS:
            raise ValueError('Temprature is too hot!')
        self._celsius = value

    def __init__(self, owner_code, contents, *, celsius, **kwargs):
        super().__init__(owner_code, contents, **kwargs)
        # Removing the temprature validation here by directly assigning the
value to property, as it will anyways do the validation.
        self.celsius = celsius

r1 = RefrigeratedShippingContainer('YML', ['Fish'], celsius=3.2)
print('r1 Temprature:', r1.celsius)
r1.celsius = -13
print('updated r1 Temprature:', r1.celsius)
# We will get the Temprature is too hot! error if it is >4 while updating t
he attribute now.
```

```
r1 Temprature: 3.2
updated r1 Temprature: -13
```

In [22]:
```
# Adding fahrenheit option to the derived class.
class ShippingContainer:
    next_serial = 1337
    @classmethod
    def _generate_serial(cls):
        result = cls.next_serial
        cls.next_serial += 1
        return result
    @staticmethod
    def _make_bic_code(owner_code, serial):
        return iso6346.create(
            owner_code=owner_code,
            serial=str(serial).zfill(6)
        )
    @classmethod
    def create_empty(cls, owner_code, **kwargs):
        return cls(owner_code, contents=[], **kwargs)
    @classmethod
    def create_with_items(cls, owner_code, items, **kwargs):
        return cls(owner_code, contents=list(items), **kwargs)
    def __init__(self, owner_code, contents, **kwargs):
        self.owner_code = owner_code
        self.contents = contents
        self.bic = self._make_bic_code(
            owner_code=owner_code,
            serial=ShippingContainer._generate_serial()
        )


class RefrigeratedShippingContainer(ShippingContainer):
    MAX_CELSIUS = 4
    @staticmethod
    def _make_bic_code(owner_code, serial):
        return iso6346.create(
            owner_code=owner_code,
            serial=str(serial).zfill(6),
            category='R'
        )
    @property
    def celsius(self):
        return self._celsius
    @celsius.setter
    def celsius(self, value):
        if value > RefrigeratedShippingContainer.MAX_CELSIUS:
            raise ValueError('Temprature is too hot!')
        self._celsius = value

    # Defining static functions here instead of normal global function as t
here functions are very specific to this derived class
    @staticmethod
    def _f_to_c(fahrenheit):
        return (fahrenheit - 32) * 5/9
    @staticmethod
    def _c_to_f(celsius):
        return celsius * 9/5 +32

    # We can get and set the temp in fahrenheit by converting it from/to ce
lsius on the fly.
    @property
    def fahrenheit(self):
        return RefrigeratedShippingContainer._c_to_f(self.celsius)
    @fahrenheit.setter
    def fahrenheit(self, value):
        self.celsius = RefrigeratedShippingContainer._f_to_c(value)

    def __init__(self, owner_code, contents, *, celsius, **kwargs):
        super().__init__(owner_code, contents, **kwargs)
        self.celsius = celsius

r1 = RefrigeratedShippingContainer('YML', ['Fish'], celsius=0)
print('r1 Temprature in fahrenheit:', r1.fahrenheit)
print('r1 Temprature in celsius:', r1.celsius)
r1.fahrenheit = 28.4
print('updated r1 Temprature in fahrenheit:', r1.fahrenheit)
print('updated r1 Temprature in celsius:', r1.celsius)
```

```
r1 Temprature in fahrenheit: 32.0
r1 Temprature in celsius: 0
updated r1 Temprature in fahrenheit: 28.4
updated r1 Temprature in celsius: -2.000000000000001
```

Properties and inheritance

In [23]:
```
# Each container has a fixed height and width but different length
# So, height and width would be class attribute while length would be insta
nce attribute
# Also defining a getter only property volume which calculates the volume o
f the container

class ShippingContainer:
    HEIGHT_FT = 8.5
    WIDTH_FT = 8.0

    next_serial = 1337
    @classmethod
    def _generate_serial(cls):
        result = cls.next_serial
        cls.next_serial += 1
        return result
    @staticmethod
    def _make_bic_code(owner_code, serial):
        return iso6346.create(
            owner_code=owner_code,
            serial=str(serial).zfill(6)
        )
    @classmethod
    def create_empty(cls, owner_code, length_ft, **kwargs):
        return cls(owner_code, length_ft, contents=[], **kwargs)
    @classmethod
    def create_with_items(cls, owner_code, length_ft, items, **kwargs):
        return cls(owner_code, length_ft, contents=list(items), **kwargs)

    @property
    def volume_ft3(self):
        return ShippingContainer.HEIGHT_FT * ShippingContainer.WIDTH_FT * s
elf.length_ft

    def __init__(self, owner_code, length_ft, contents, **kwargs):
        self.owner_code = owner_code
        self.contents = contents
        self.length_ft = length_ft
        self.bic = self._make_bic_code(
            owner_code=owner_code,
            serial=ShippingContainer._generate_serial()
        )


class RefrigeratedShippingContainer(ShippingContainer):
    MAX_CELSIUS = 4
    @staticmethod
    def _make_bic_code(owner_code, serial):
        return iso6346.create(
            owner_code=owner_code,
            serial=str(serial).zfill(6),
            category='R'
        )
    @property
    def celsius(self):
        return self._celsius
    @celsius.setter
    def celsius(self, value):
        if value > RefrigeratedShippingContainer.MAX_CELSIUS:
            raise ValueError('Temprature is too hot!')
        self._celsius = value
    @staticmethod
    def _f_to_c(fahrenheit):
        return (fahrenheit - 32) * 5/9
    @staticmethod
    def _c_to_f(celsius):
        return celsius * 9/5 +32
    @property
    def fahrenheit(self):
        return RefrigeratedShippingContainer._c_to_f(self.celsius)
    @fahrenheit.setter
    def fahrenheit(self, value):
        self.celsius = RefrigeratedShippingContainer._f_to_c(value)
    def __init__(self, owner_code, contents, *, celsius, **kwargs):
        super().__init__(owner_code, contents, **kwargs)
        self.celsius = celsius

c1 = ShippingContainer.create_empty('YML', length_ft=20)
print('volume of an empty 20ft length container:', c1.volume_ft3)
```

```
volume of an empty 20ft length container: 1360.0
```

In [24]:
```
# For refrigerated derived class we will have to specify the 100 sqft space
which is taken by the cooling machine.
# To override property getter we will simply have to redefine in derived cl
ass.
class ShippingContainer:
    HEIGHT_FT = 8.5
    WIDTH_FT = 8.0

    next_serial = 1337
    @classmethod
    def _generate_serial(cls):
        result = cls.next_serial
        cls.next_serial += 1
        return result
    @staticmethod
    def _make_bic_code(owner_code, serial):
        return iso6346.create(
            owner_code=owner_code,
            serial=str(serial).zfill(6)
        )
    @classmethod
    def create_empty(cls, owner_code, length_ft, **kwargs):
        return cls(owner_code, length_ft, contents=[], **kwargs)
    @classmethod
    def create_with_items(cls, owner_code, length_ft, items, **kwargs):
        return cls(owner_code, length_ft, contents=list(items), **kwargs)

    @property
    def volume_ft3(self):
        return ShippingContainer.HEIGHT_FT * ShippingContainer.WIDTH_FT * s
elf.length_ft

    def __init__(self, owner_code, length_ft, contents, **kwargs):
        self.owner_code = owner_code
        self.contents = contents
        self.length_ft = length_ft
        self.bic = self._make_bic_code(
            owner_code=owner_code,
            serial=ShippingContainer._generate_serial()
        )


class RefrigeratedShippingContainer(ShippingContainer):
    FRIDGE_VOLUME_FT3 = 100

    MAX_CELSIUS = 4
    @staticmethod
    def _make_bic_code(owner_code, serial):
        return iso6346.create(
            owner_code=owner_code,
            serial=str(serial).zfill(6),
            category='R'
        )
    @property
    def celsius(self):
        return self._celsius
    @celsius.setter
    def celsius(self, value):
        if value > RefrigeratedShippingContainer.MAX_CELSIUS:
            raise ValueError('Temprature is too hot!')
        self._celsius = value
    @staticmethod
    def _f_to_c(fahrenheit):
        return (fahrenheit - 32) * 5/9
    @staticmethod
    def _c_to_f(celsius):
        return celsius * 9/5 +32
    @property
    def fahrenheit(self):
        return RefrigeratedShippingContainer._c_to_f(self.celsius)
    @fahrenheit.setter
    def fahrenheit(self, value):
        self.celsius = RefrigeratedShippingContainer._f_to_c(value)

    # Call the parent/super property and then do the further needed modific
ation
    @property
    def volume_ft3(self):
        return super().volume_ft3 - RefrigeratedShippingContainer.FRIDGE_VO
```

```
LUME_FT3

    def __init__(self, owner_code, length_ft, contents, *, celsius, **kwarg
s):
        super().__init__(owner_code, length_ft, contents, **kwargs)
        self.celsius = celsius

r1 = RefrigeratedShippingContainer.create_empty('YML', length_ft=20, celsiu
s=2)
print('volume of an empty 20ft length refrigerated container:', r1.volume_f
t3)
```

```
volume of an empty 20ft length refrigerated container: 1260.0
```

In [25]:

```
# Overriding a setter is much more complicated
# Lets define a heated container which also has a lower limit for tempratur
e.

class HeatedRefrigeratedShippingContainer(RefrigeratedShippingContainer):

    MIN_CELSIUS = -20

    # @celsisus.setter will throw error as celsius property is not in bound
    @RefrigeratedShippingContainer.celsius.setter
    def celsius(self, value):
        if value < HeatedRefrigeratedShippingContainer.MIN_CELSIUS:
            raise ValueError('Temprature too cold!')
        # super().celsius = value doesn't work for some reason
        # Setting celsius through RefrigeratedShippingContainer will check
 for the upper limit condition also
        RefrigeratedShippingContainer.celsius.fset(self, value)
```

As we can see overriding these setters can get messy.
We can use template method to override properties instead
According to which we should never override properties, we should always delegate them to regular methods and then override them instead.

In [26]:
```
# We will override the method instead of properties
class ShippingContainer:
    HEIGHT_FT = 8.5
    WIDTH_FT = 8.0

    next_serial = 1337
    @classmethod
    def _generate_serial(cls):
        result = cls.next_serial
        cls.next_serial += 1
        return result
    @staticmethod
    def _make_bic_code(owner_code, serial):
        return iso6346.create(
            owner_code=owner_code,
            serial=str(serial).zfill(6)
        )
    @classmethod
    def create_empty(cls, owner_code, length_ft, **kwargs):
        return cls(owner_code, length_ft, contents=[], **kwargs)
    @classmethod
    def create_with_items(cls, owner_code, length_ft, items, **kwargs):
        return cls(owner_code, length_ft, contents=list(items), **kwargs)

    @property
    def volume_ft3(self):
        return self._calc_volume()

    def _calc_volume(self):
        return ShippingContainer.HEIGHT_FT * ShippingContainer.WIDTH_FT * s
elf.length_ft

    def __init__(self, owner_code, length_ft, contents, **kwargs):
        self.owner_code = owner_code
        self.contents = contents
        self.length_ft = length_ft
        self.bic = self._make_bic_code(
            owner_code=owner_code,
            serial=ShippingContainer._generate_serial()
        )


class RefrigeratedShippingContainer(ShippingContainer):
    FRIDGE_VOLUME_FT3 = 100

    MAX_CELSIUS = 4
    @staticmethod
    def _make_bic_code(owner_code, serial):
        return iso6346.create(
            owner_code=owner_code,
            serial=str(serial).zfill(6),
            category='R'
        )
    @property
    def celsius(self):
        return self._celsius
    @celsius.setter
    def celsius(self, value):
        return self._set_celsius(value)

    # Making normal method to override rather than to override setter
    def _set_celsius(self, value):
        if value > RefrigeratedShippingContainer.MAX_CELSIUS:
            raise ValueError('Temprature is too hot!')
        self._celsius = value

    @staticmethod
    def _f_to_c(fahrenheit):
        return (fahrenheit - 32) * 5/9
    @staticmethod
    def _c_to_f(celsius):
        return celsius * 9/5 +32
    @property
    def fahrenheit(self):
        return RefrigeratedShippingContainer._c_to_f(self.celsius)
    @fahrenheit.setter
    def fahrenheit(self, value):
        self.celsius = RefrigeratedShippingContainer._f_to_c(value)

    # Overriding method instead of property
    def _calc_volume(self):
        return super()._calc_volume() - RefrigeratedShippingContainer.FRIDG
E_VOLUME_FT3

    def __init__(self, owner_code, length_ft, contents, *, celsius, **kwarg
s):
        super().__init__(owner_code, length_ft, contents, **kwargs)
        self.celsius = celsius

class HeatedRefrigeratedShippingContainer(RefrigeratedShippingContainer):
    MIN_CELSIUS = -20

    def _set_celsius(self, value):
        if value < HeatedRefrigeratedShippingContainer.MIN_CELSIUS:
            raise ValueError('Temprature too cold!')
        super()._set_celsius(value)
```

In [27]:
```
c1 = ShippingContainer.create_empty('YML', length_ft=20)
print('volume of an empty 20ft length container:', c1.volume_ft3)
r1 = RefrigeratedShippingContainer.create_empty('YML', length_ft=20, celsiu
s=2)
print('volume of an empty 20ft length refrigerated container:', r1.volume_f
t3)
```

```
volume of an empty 20ft length container: 1360.0
volume of an empty 20ft length refrigerated container: 1260.0
```

In [28]:
```
try:
    r1 = HeatedRefrigeratedShippingContainer.create_empty('YML', length_ft=
20, celsius=5)
except ValueError as e:
    print(e)
try:
    r1 = HeatedRefrigeratedShippingContainer.create_empty('YML', length_ft=
20, celsius=-25)
except ValueError as e:
    print(e)
```

```
Temprature is too hot!
Temprature too cold!
```