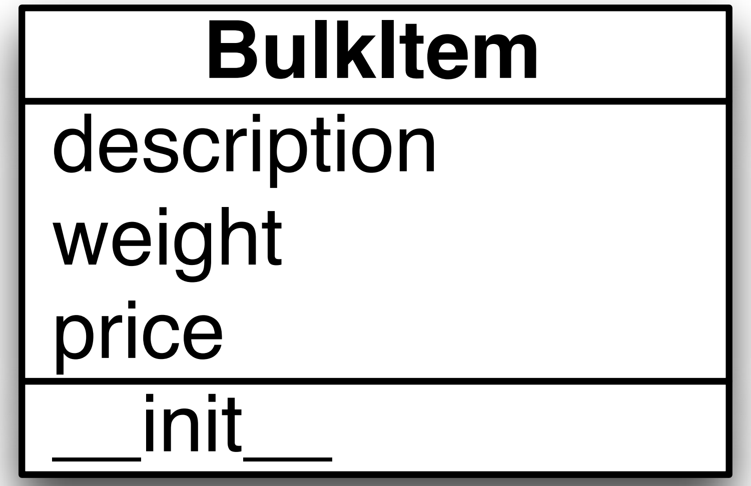
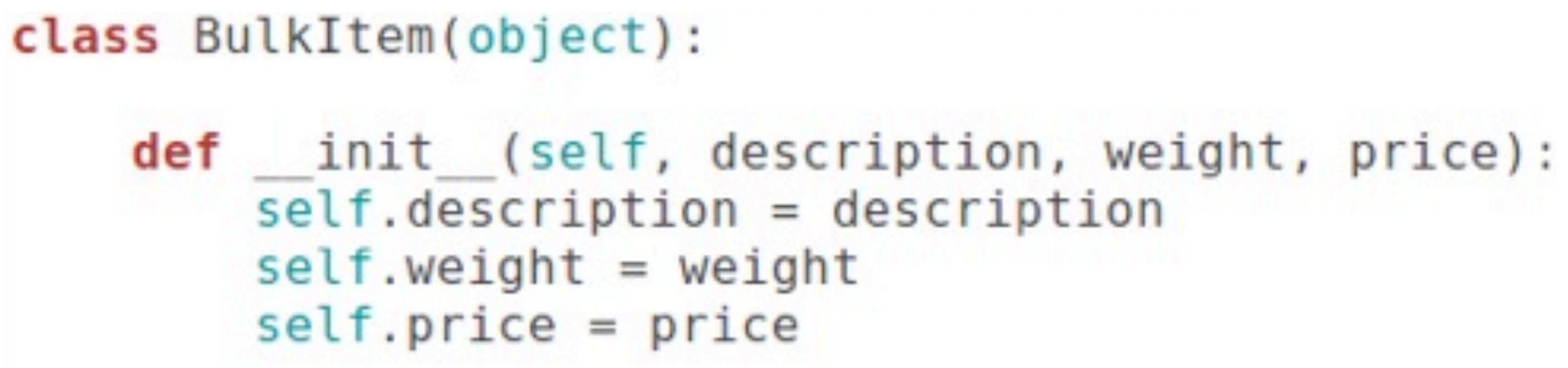
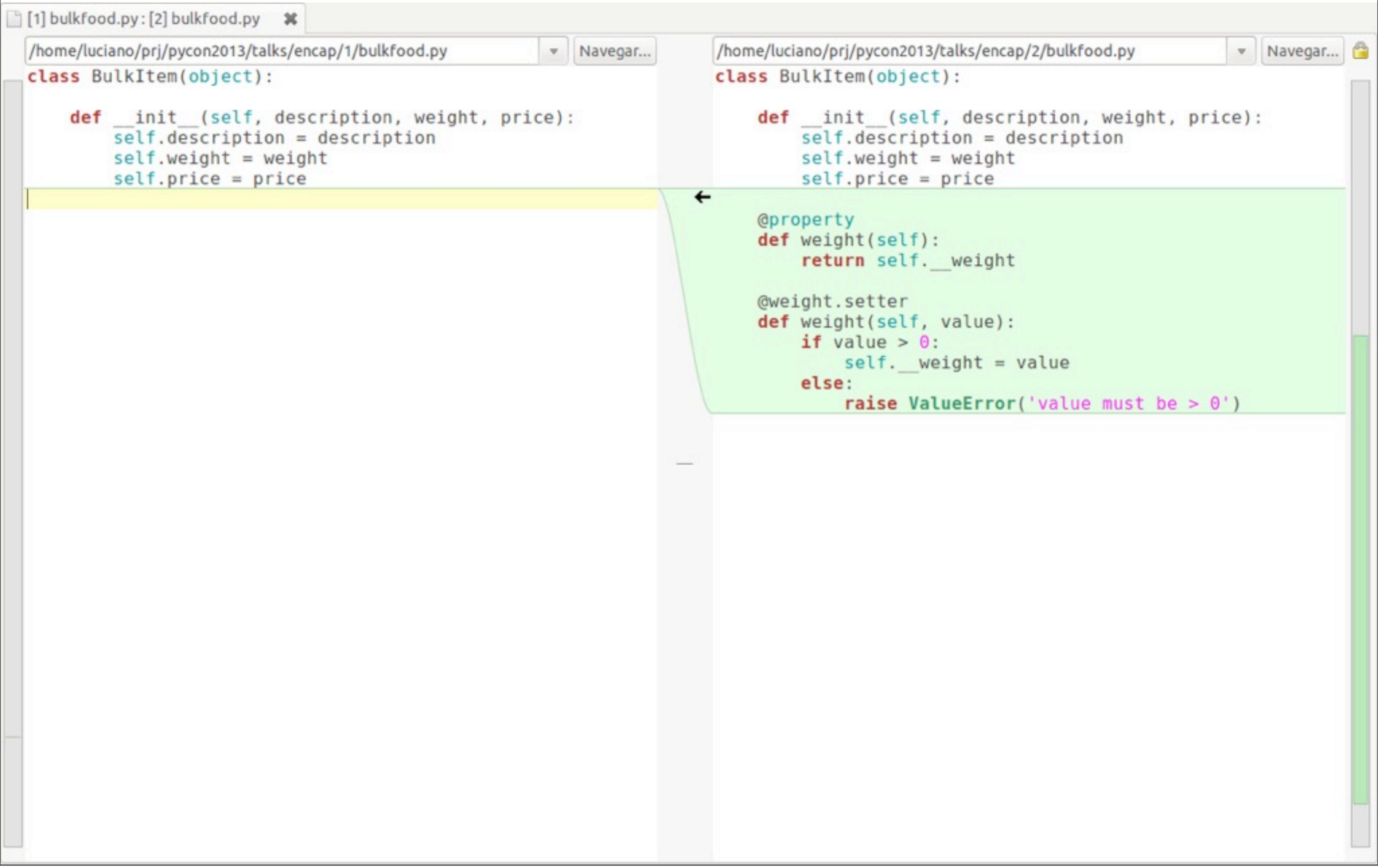
❶ the simplest thing





```
class BulkItem(object):

    def __init__(self, description, weight, price):
        self.description = description
        self.weight = weight
        self.price = price
```

❷ validation via property



[1] bulkfood.py : [2] bulkfood.py

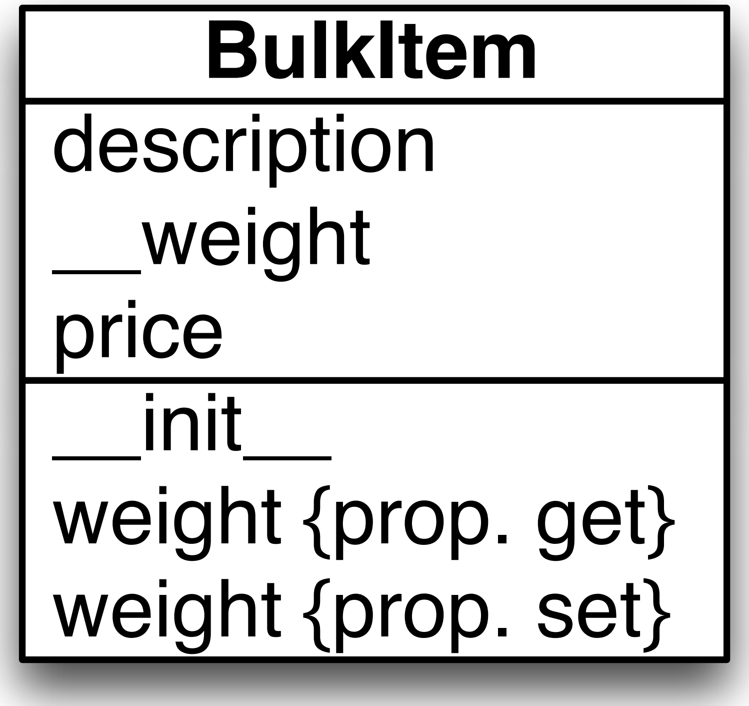
/home/luciano/prj/pycon2013/talks/encap/1/bulkfood.py

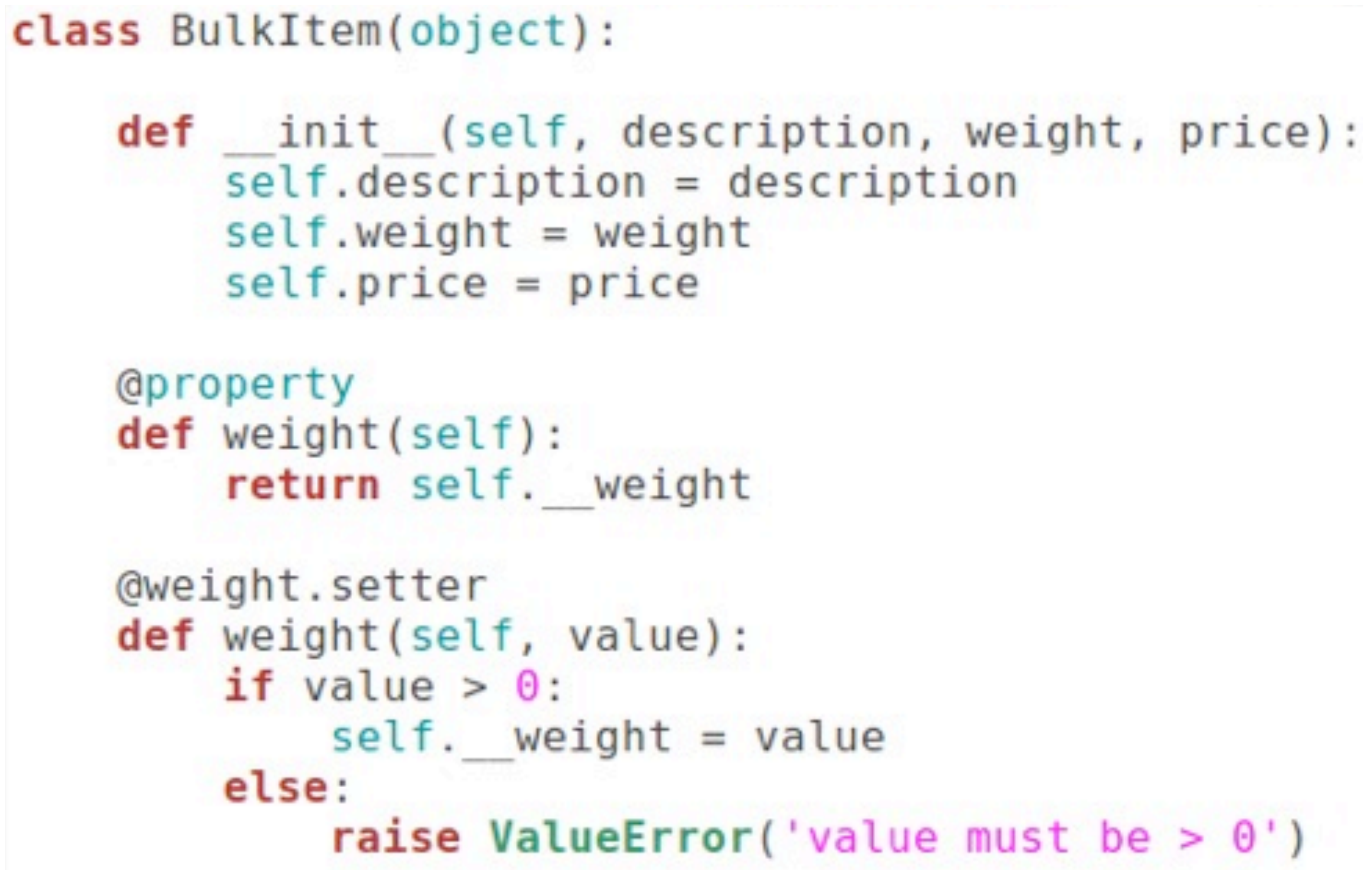
```python
class BulkItem(object):

    def __init__(self, description, weight, price):
        self.description = description
        self.weight = weight
        self.price = price
```

/home/luciano/prj/pycon2013/talks/encap/2/bulkfood.py

```python
class BulkItem(object):

    def __init__(self, description, weight, price):
        self.description = description
        self.weight = weight
        self.price = price

    @property
    def weight(self):
        return self.__weight

    @weight.setter
    def weight(self, value):
        if value > 0:
            self.__weight = value
        else:
            raise ValueError('value must be > 0')
```

❷ validation via property

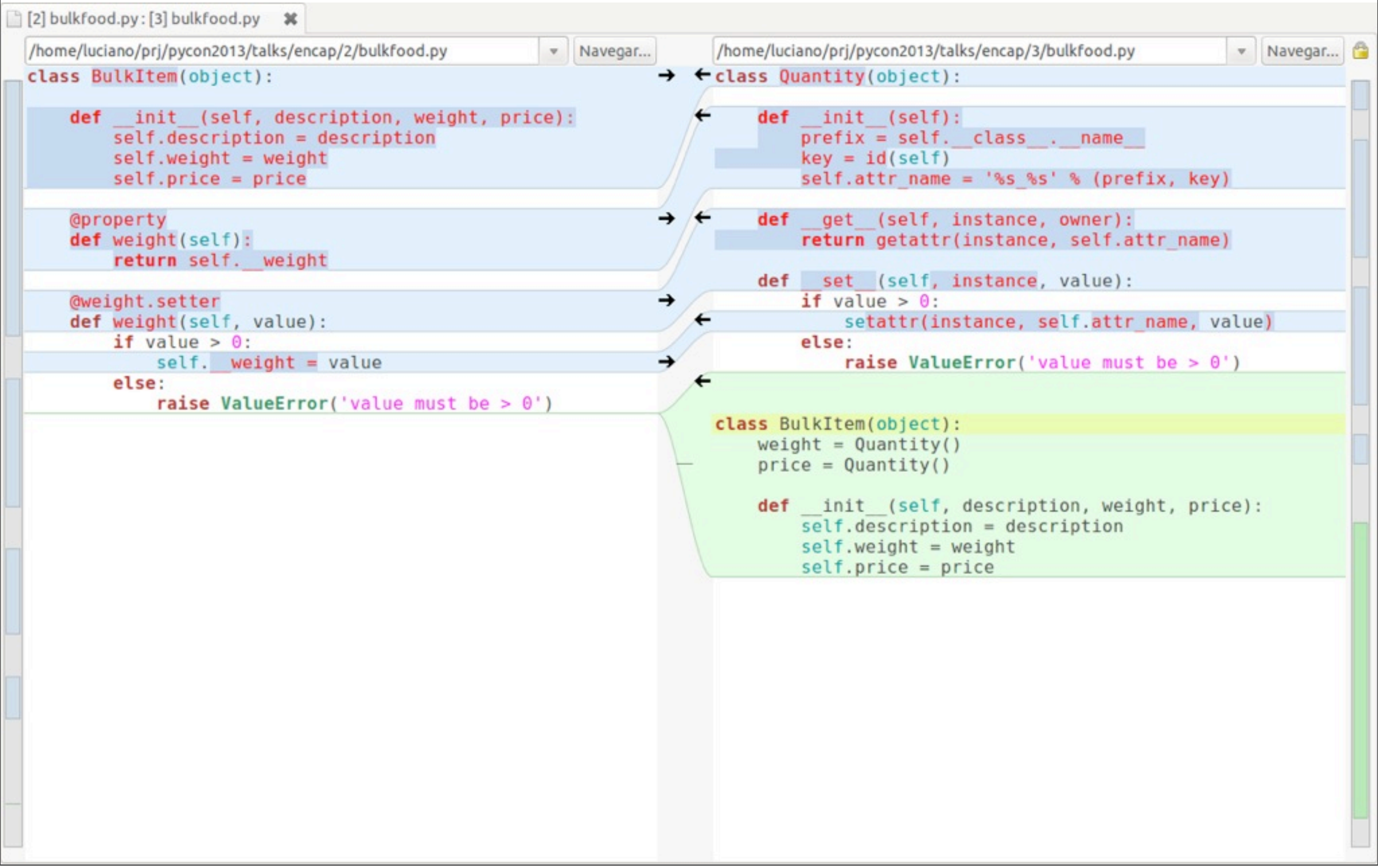




```
class BulkItem(object):

    def __init__(self, description, weight, price):
        self.description = description
        self.weight = weight
        self.price = price

    @property
    def weight(self):
        return self.__weight

    @weight.setter
    def weight(self, value):
        if value > 0:
            self.__weight = value
        else:
            raise ValueError('value must be > 0')
```

This works, but what if **price** needs a similar treatment?

❸ validation via descriptors



[2] bulkfood.py : [3] bulkfood.py

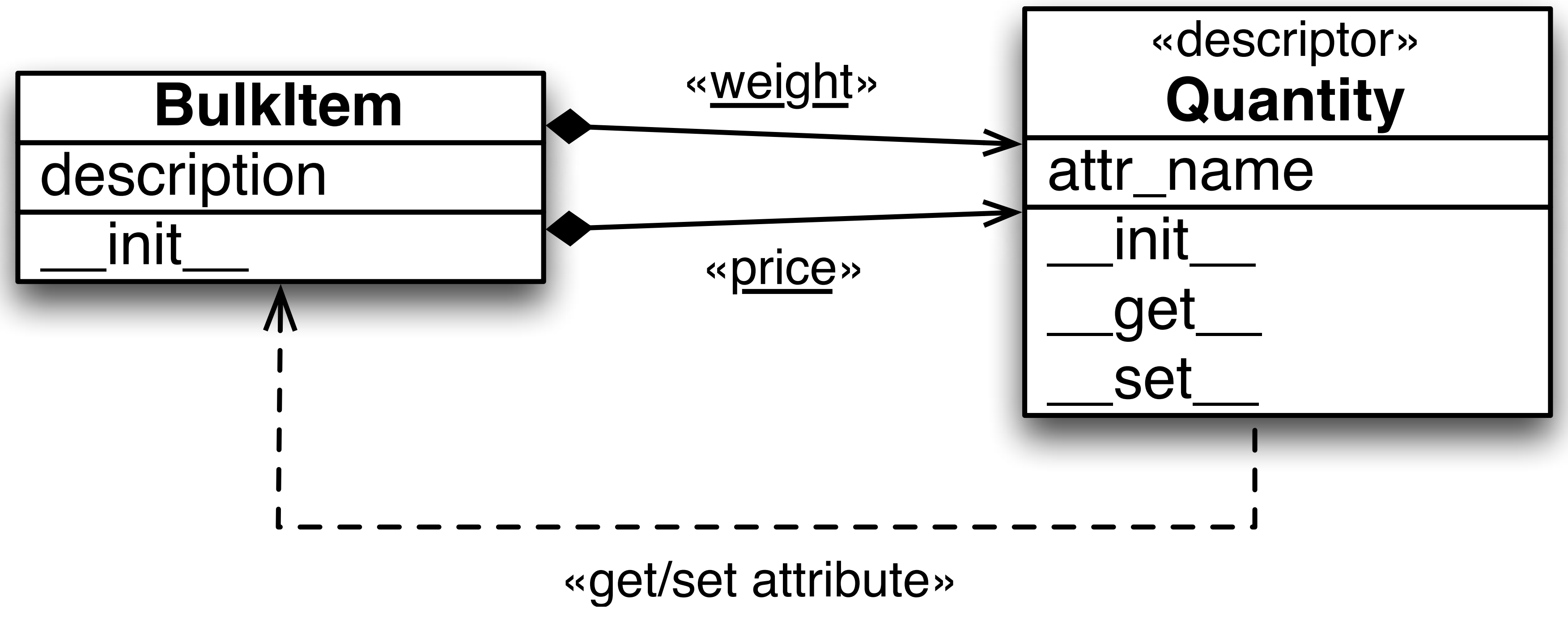
/home/luciano/prj/pycon2013/talks/encap/2/bulkfood.py

```python
class BulkItem(object):

    def __init__(self, description, weight, price):
        self.description = description
        self.weight = weight
        self.price = price

    @property
    def weight(self):
        return self.__weight

    @weight.setter
    def weight(self, value):
        if value > 0:
            self.__weight = value
        else:
            raise ValueError('value must be > 0')
```

/home/luciano/prj/pycon2013/talks/encap/3/bulkfood.py

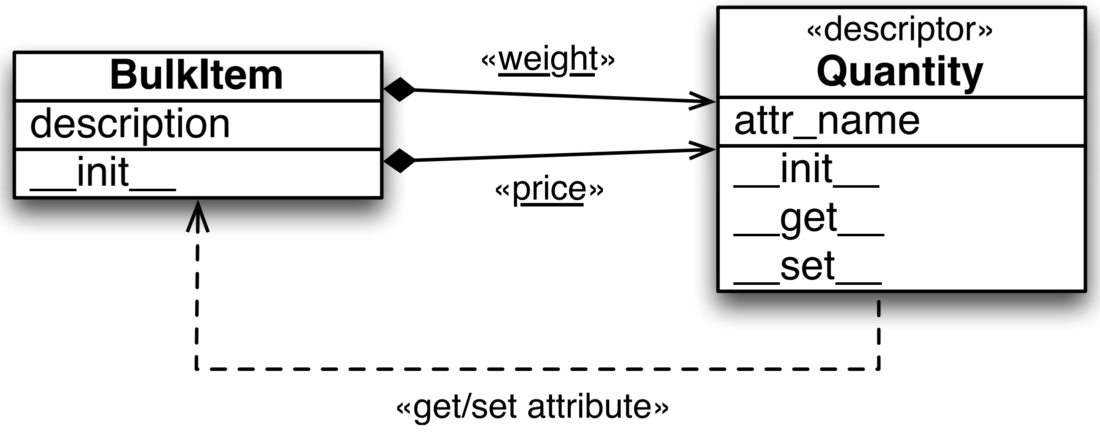
```python
class Quantity(object):

    def __init__(self):
        prefix = self.__class__.__name__
        key = id(self)
        self.attr_name = '%s_%s' % (prefix, key)

    def __get__(self, instance, owner):
        return getattr(instance, self.attr_name)

    def __set__(self, instance, value):
        if value > 0:
            setattr(instance, self.attr_name, value)
        else:
            raise ValueError('value must be > 0')


class BulkItem(object):
    weight = Quantity()
    price = Quantity()

    def __init__(self, description, weight, price):
        self.description = description
        self.weight = weight
        self.price = price
```

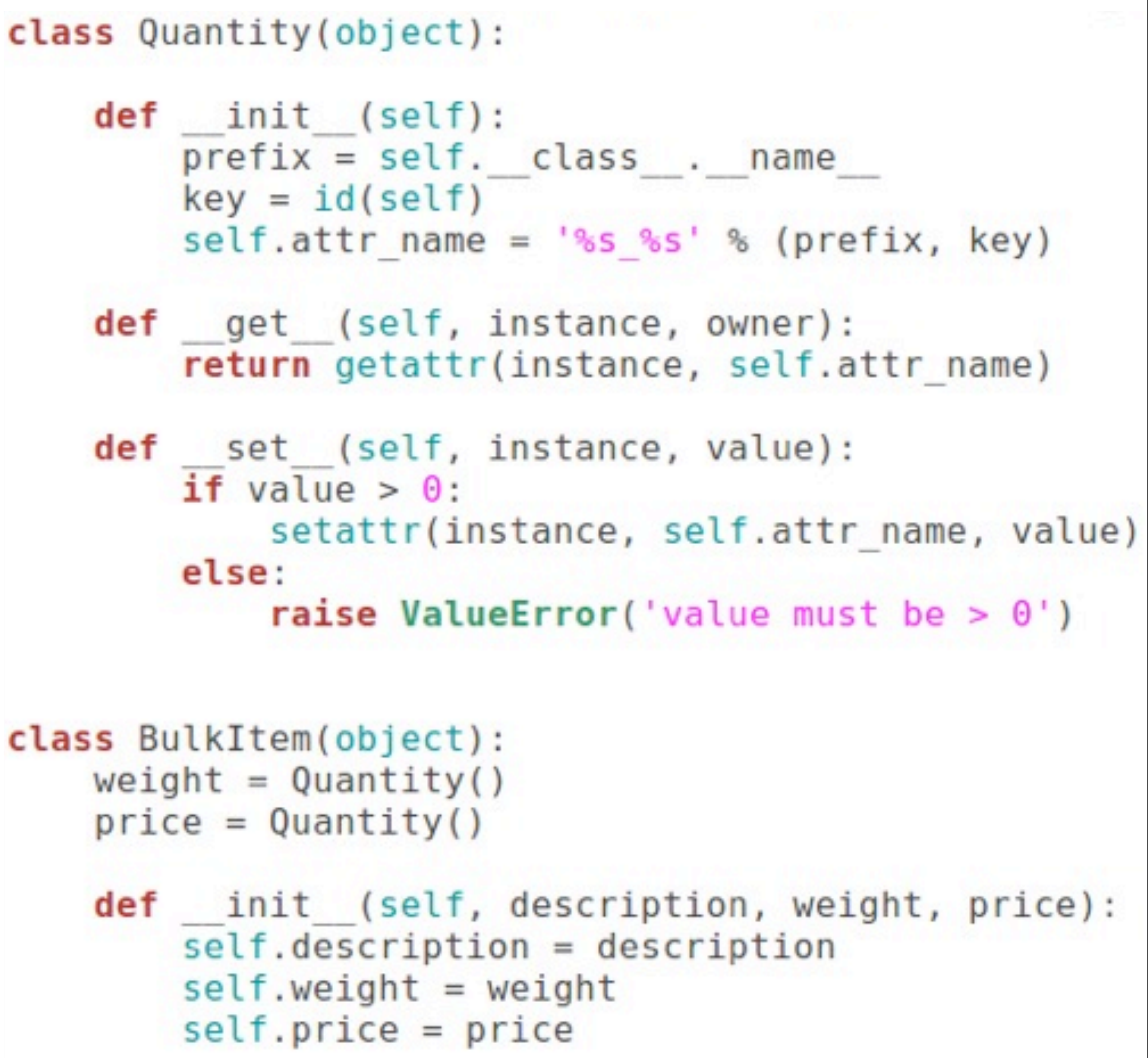
❸ validation via descriptors



Descriptors enable reuse of validation logic through **composition**

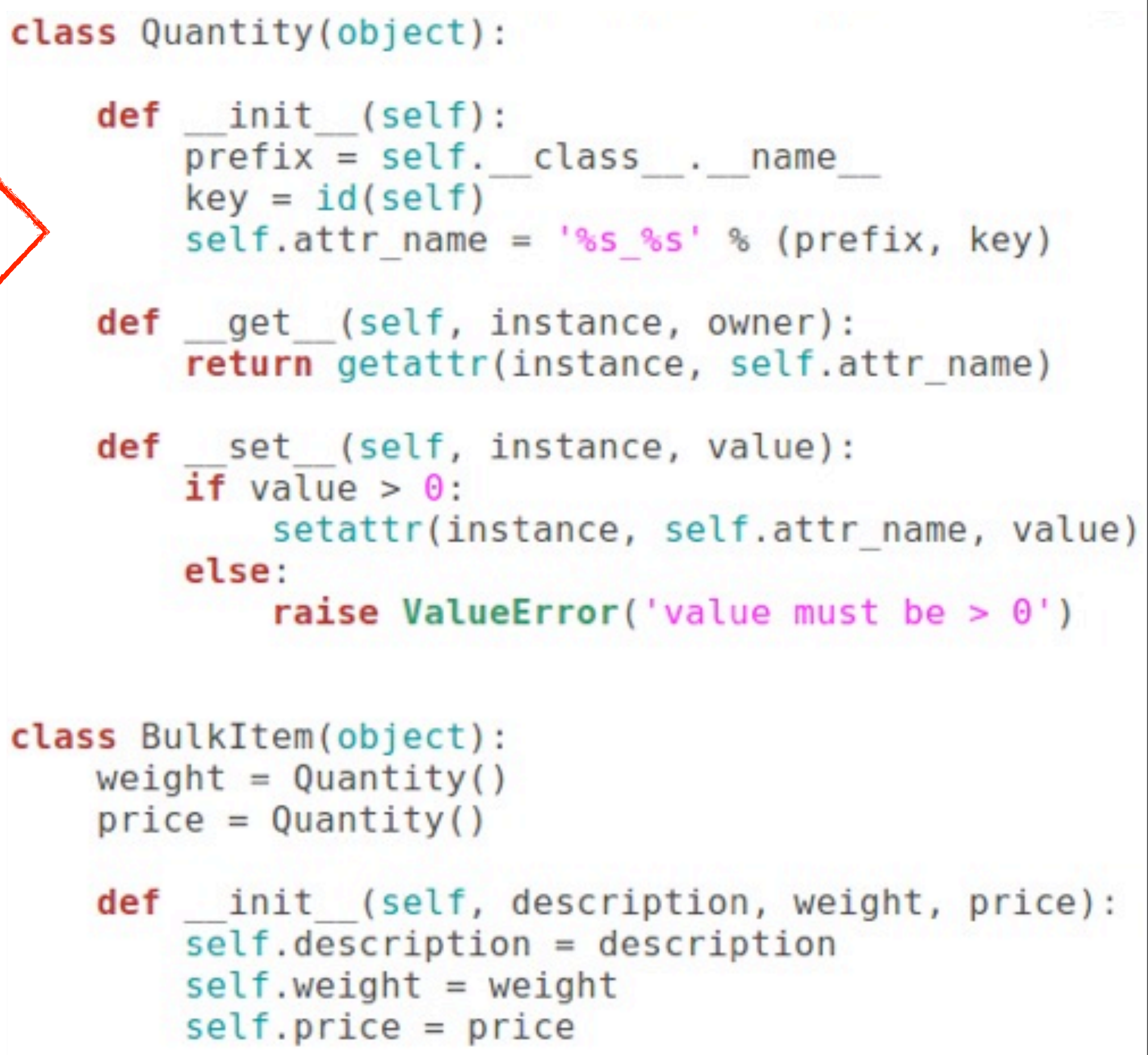
❸ validation via descriptors





```
class Quantity(object):

    def __init__(self):
        prefix = self.__class__.__name__
        key = id(self)
        self.attr_name = '%s_%s' % (prefix, key)

    def __get__(self, instance, owner):
        return getattr(instance, self.attr_name)

    def __set__(self, instance, value):
        if value > 0:
            setattr(instance, self.attr_name, value)
        else:
            raise ValueError('value must be > 0')


class BulkItem(object):
    weight = Quantity()
    price = Quantity()

    def __init__(self, description, weight, price):
        self.description = description
        self.weight = weight
        self.price = price
```

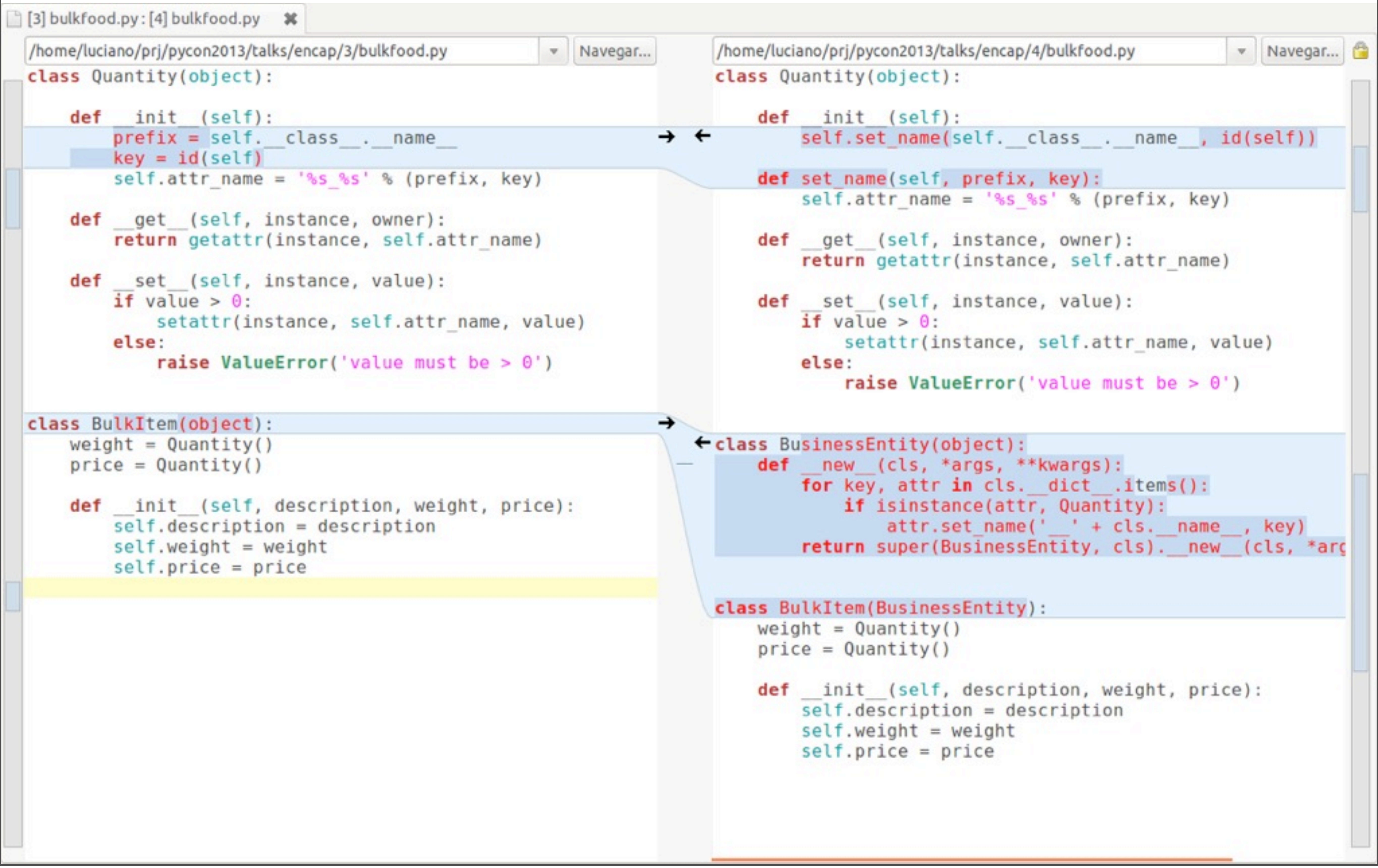
❸ validation via descriptors



```
class Quantity(object):

    def __init__(self):
        prefix = self.__class__.__name__
        key = id(self)
        self.attr_name = '%s_%s' % (prefix, key)

    def __get__(self, instance, owner):
        return getattr(instance, self.attr_name)

    def __set__(self, instance, value):
        if value > 0:
            setattr(instance, self.attr_name, value)
        else:
            raise ValueError('value must be > 0')


class BulkItem(object):
    weight = Quantity()
    price = Quantity()

    def __init__(self, description, weight, price):
        self.description = description
        self.weight = weight
        self.price = price
```

Field data is stored in **BulkItem** instance attributes with generic names like **Quantity_14199423**

❹ proper names for attrs



[3] bulkfood.py : [4] bulkfood.py

/home/luciano/prj/pycon2013/talks/encap/3/bulkfood.py

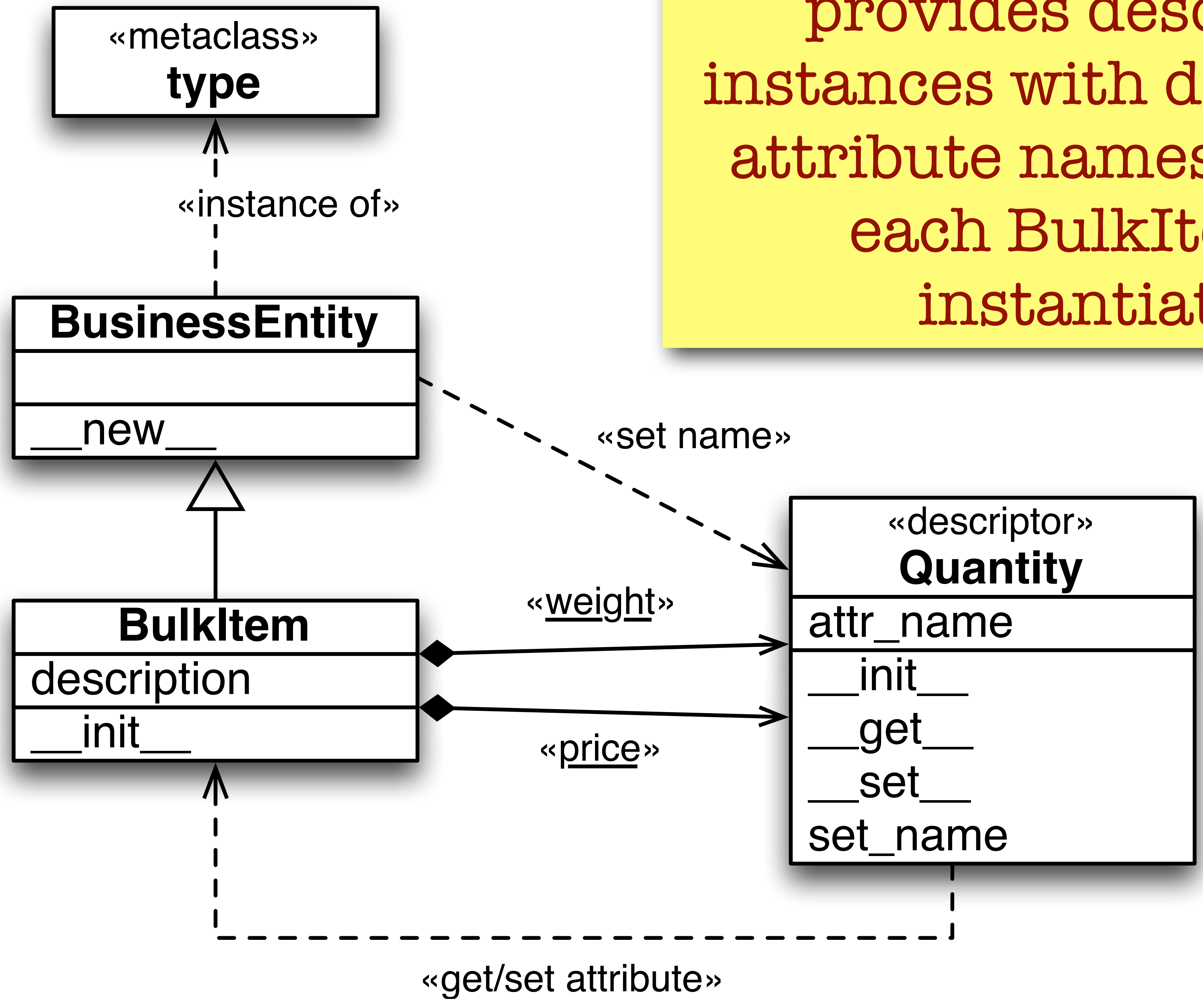
```python
class Quantity(object):

    def __init__(self):
        prefix = self.__class__.__name__
        key = id(self)
        self.attr_name = '%s_%s' % (prefix, key)

    def __get__(self, instance, owner):
        return getattr(instance, self.attr_name)

    def __set__(self, instance, value):
        if value > 0:
            setattr(instance, self.attr_name, value)
        else:
            raise ValueError('value must be > 0')


class BulkItem(object):
    weight = Quantity()
    price = Quantity()

    def __init__(self, description, weight, price):
        self.description = description
        self.weight = weight
        self.price = price
```

/home/luciano/prj/pycon2013/talks/encap/4/bulkfood.py

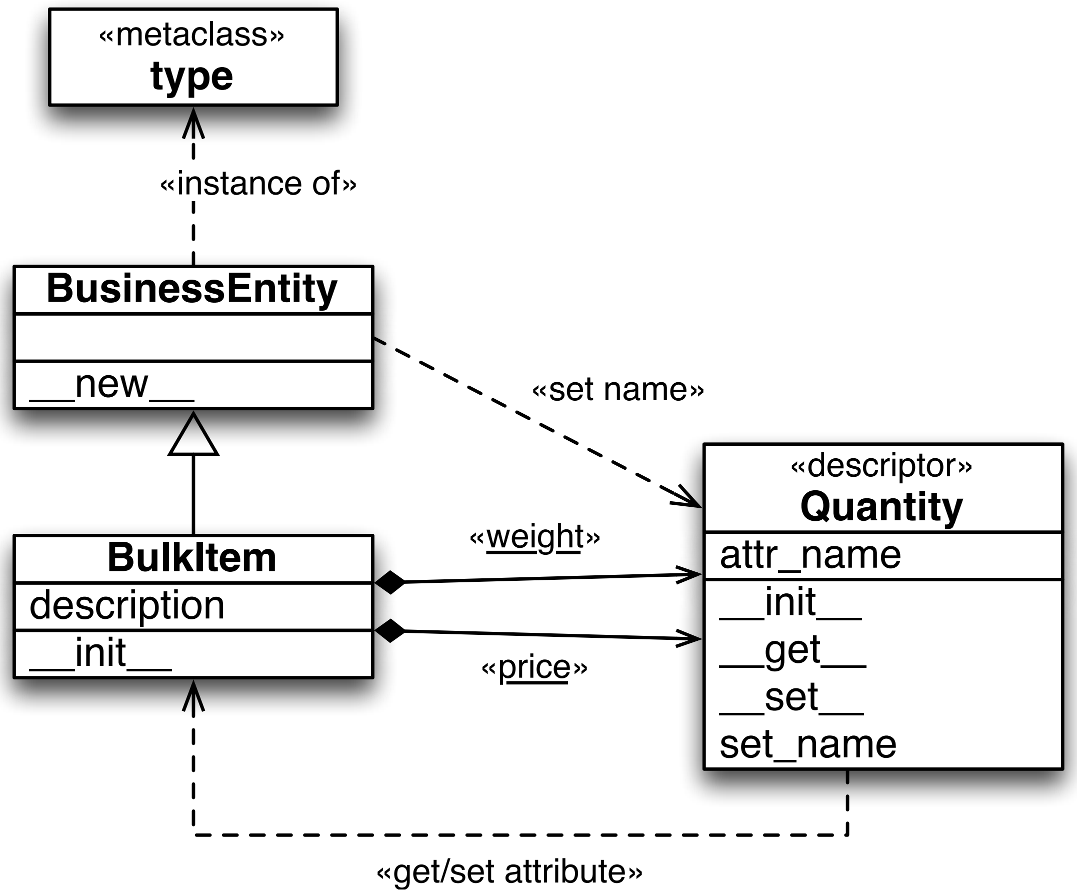
```python
class Quantity(object):

    def __init__(self):
        self.set_name(self.__class__.__name__, id(self))

    def set_name(self, prefix, key):
        self.attr_name = '%s_%s' % (prefix, key)

    def __get__(self, instance, owner):
        return getattr(instance, self.attr_name)

    def __set__(self, instance, value):
        if value > 0:
            setattr(instance, self.attr_name, value)
        else:
            raise ValueError('value must be > 0')


class BusinessEntity(object):
    def __new__(cls, *args, **kwargs):
        for key, attr in cls.__dict__.items():
            if isinstance(attr, Quantity):
                attr.set_name('__' + cls.__name__, key)
        return super(BusinessEntity, cls).__new__(cls, *arg


class BulkItem(BusinessEntity):
    weight = Quantity()
    price = Quantity()

    def __init__(self, description, weight, price):
        self.description = description
        self.weight = weight
        self.price = price
```

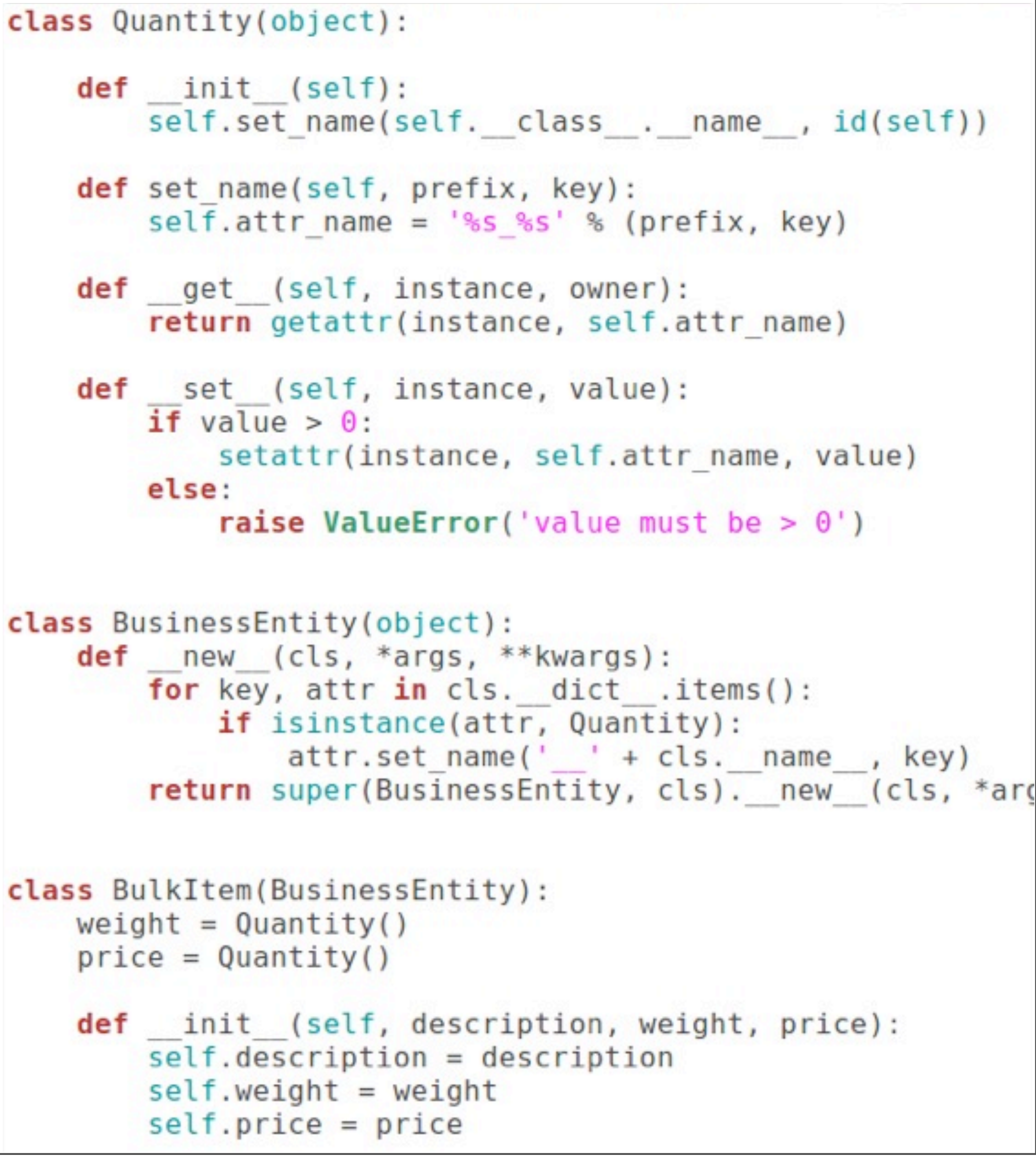
❹ proper names for attrs

BusinessEntity.__new__ provides descriptor instances with descriptive attribute names at when each BulkItem is instantiated



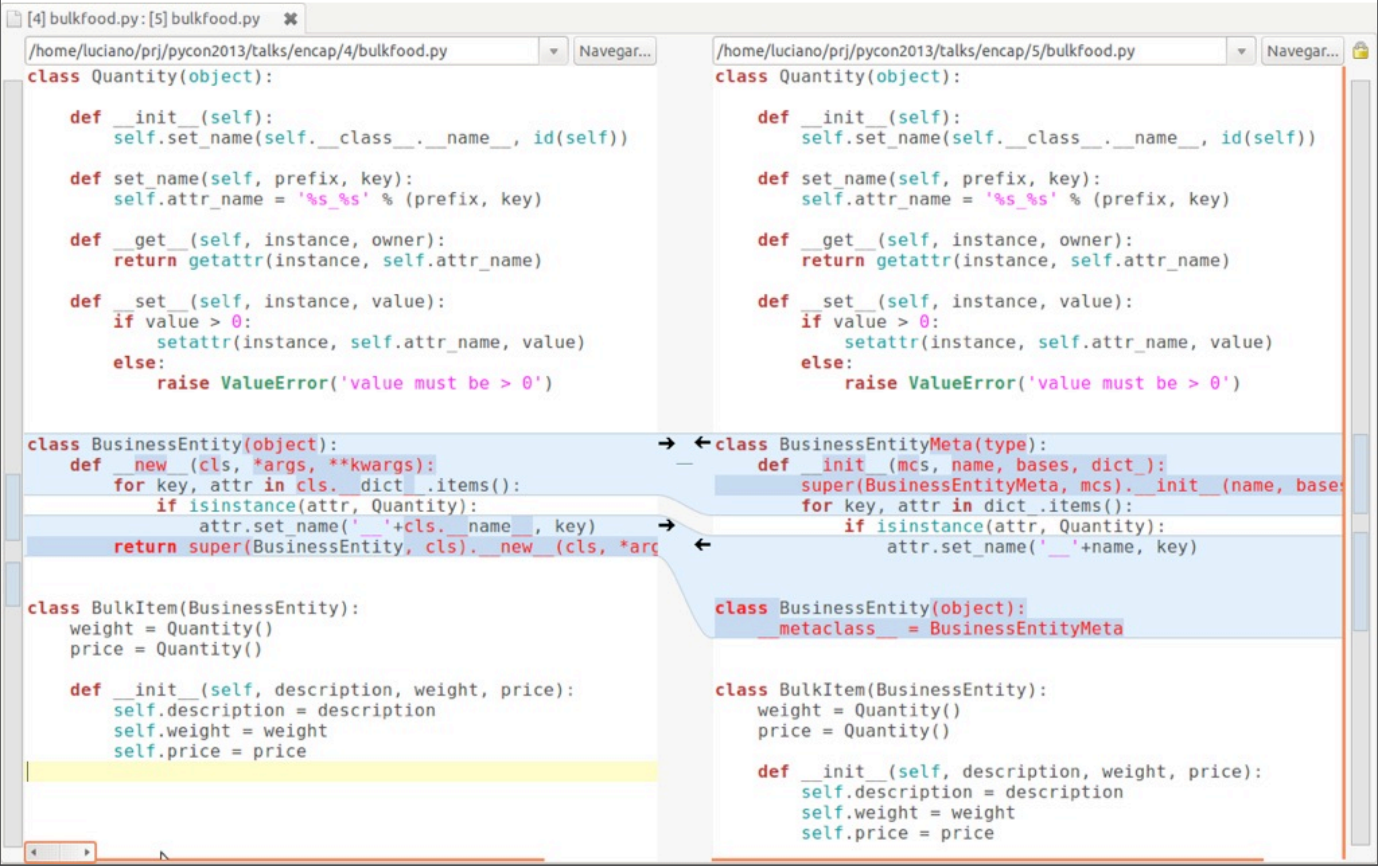
❹ proper names for attrs





```
class Quantity(object):

    def __init__(self):
        self.set_name(self.__class__.__name__, id(self))

    def set_name(self, prefix, key):
        self.attr_name = '%s_%s' % (prefix, key)

    def __get__(self, instance, owner):
        return getattr(instance, self.attr_name)

    def __set__(self, instance, value):
        if value > 0:
            setattr(instance, self.attr_name, value)
        else:
            raise ValueError('value must be > 0')


class BusinessEntity(object):
    def __new__(cls, *args, **kwargs):
        for key, attr in cls.__dict__.items():
            if isinstance(attr, Quantity):
                attr.set_name('__' + cls.__name__, key)
        return super(BusinessEntity, cls).__new__(cls, *arg


class BulkItem(BusinessEntity):
    weight = Quantity()
    price = Quantity()

    def __init__(self, description, weight, price):
        self.description = description
        self.weight = weight
        self.price = price
```

❺ avoiding redundant work



[4] bulkfood.py : [5] bulkfood.py

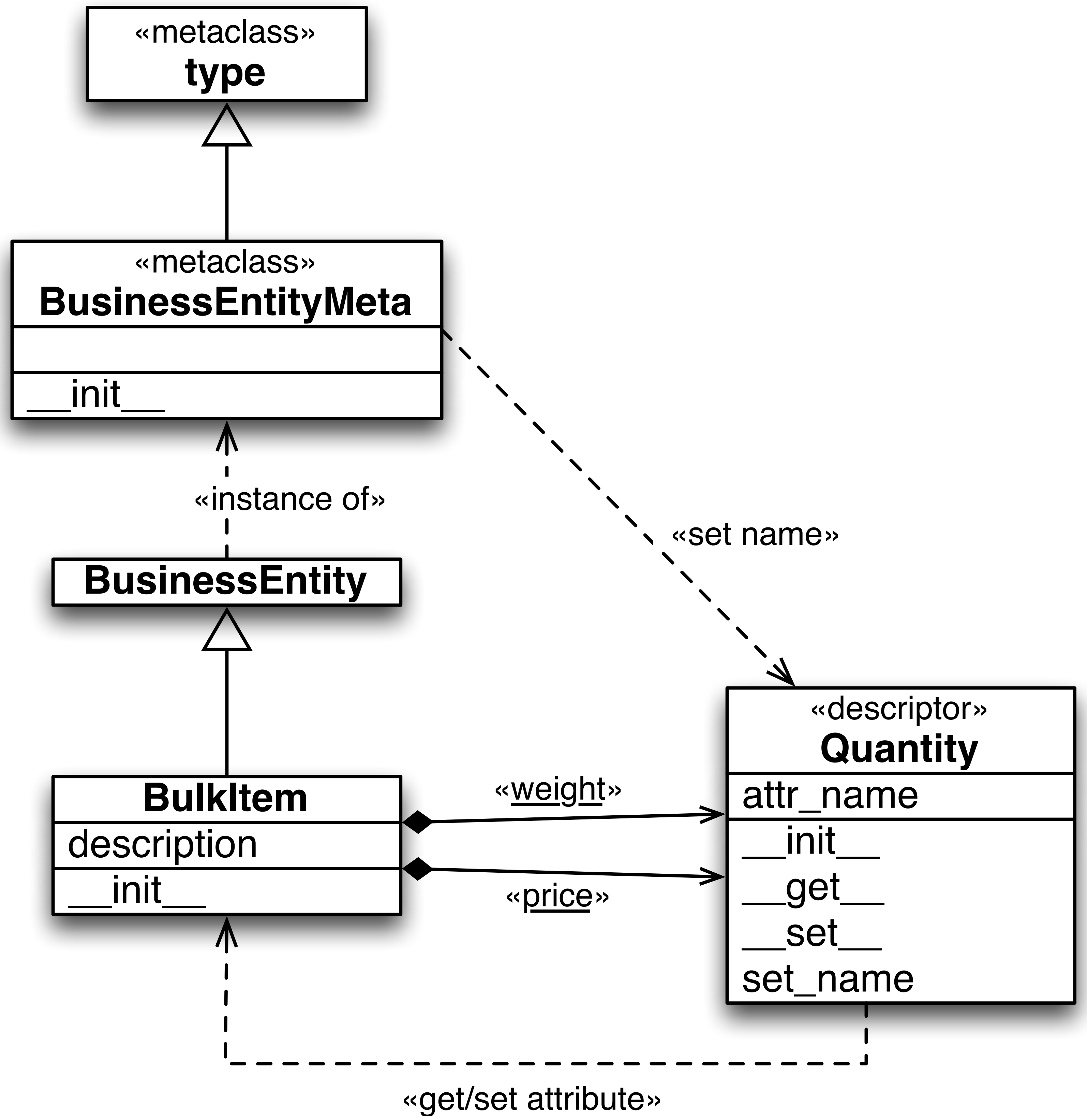
/home/luciano/prj/pycon2013/talks/encap/4/bulkfood.py

```python
class Quantity(object):

    def __init__(self):
        self.set_name(self.__class__.__name__, id(self))

    def set_name(self, prefix, key):
        self.attr_name = '%s_%s' % (prefix, key)

    def __get__(self, instance, owner):
        return getattr(instance, self.attr_name)

    def __set__(self, instance, value):
        if value > 0:
            setattr(instance, self.attr_name, value)
        else:
            raise ValueError('value must be > 0')


class BusinessEntity(object):
    def __new__(cls, *args, **kwargs):
        for key, attr in cls.__dict__.items():
            if isinstance(attr, Quantity):
                attr.set_name('__'+cls.__name__, key)
        return super(BusinessEntity, cls).__new__(cls, *arg


class BulkItem(BusinessEntity):
    weight = Quantity()
    price = Quantity()

    def __init__(self, description, weight, price):
        self.description = description
        self.weight = weight
        self.price = price
```

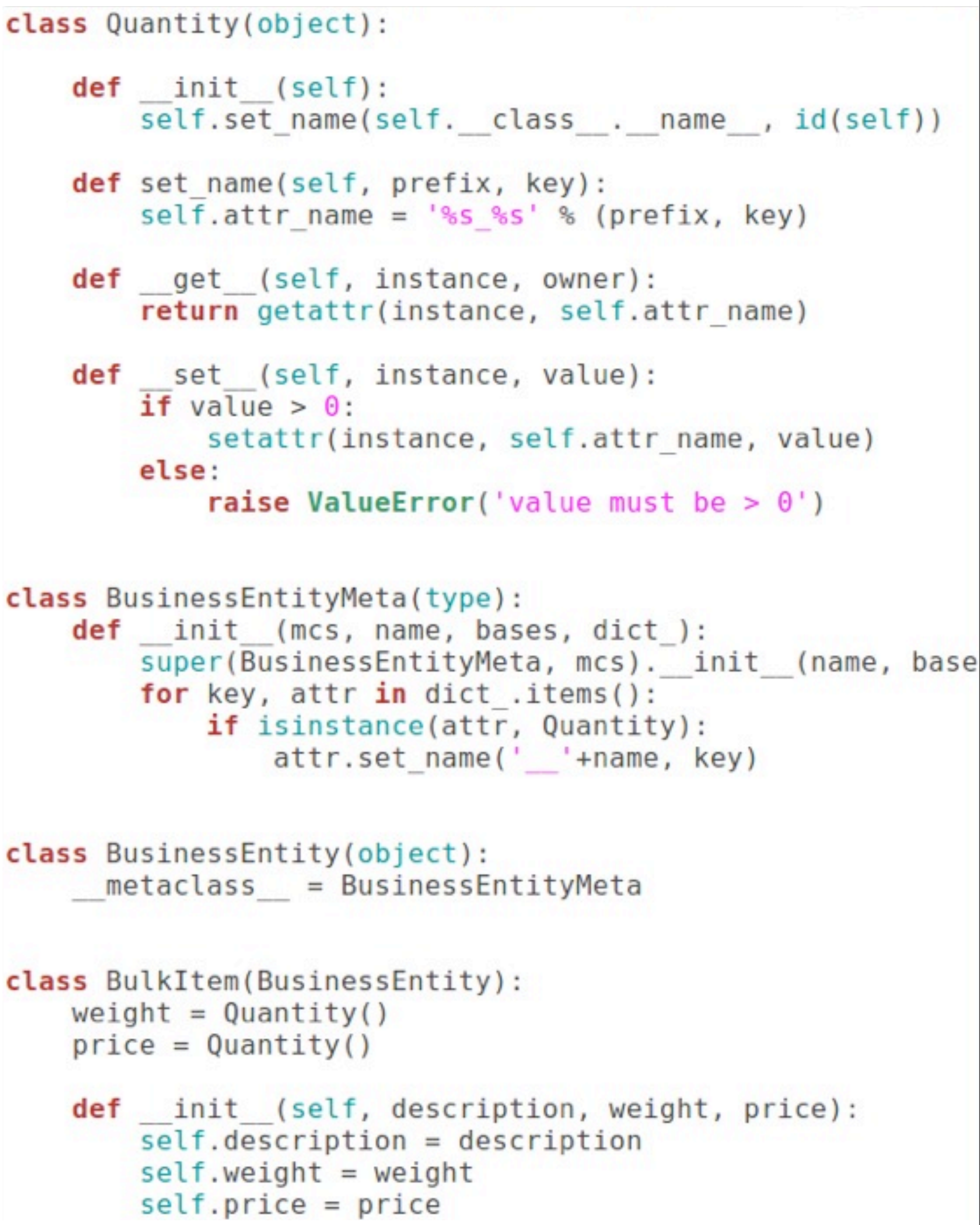
/home/luciano/prj/pycon2013/talks/encap/5/bulkfood.py

```python
class Quantity(object):

    def __init__(self):
        self.set_name(self.__class__.__name__, id(self))

    def set_name(self, prefix, key):
        self.attr_name = '%s_%s' % (prefix, key)

    def __get__(self, instance, owner):
        return getattr(instance, self.attr_name)

    def __set__(self, instance, value):
        if value > 0:
            setattr(instance, self.attr_name, value)
        else:
            raise ValueError('value must be > 0')


class BusinessEntityMeta(type):
    def __init__(mcs, name, bases, dict_):
        super(BusinessEntityMeta, mcs).__init__(name, base
        for key, attr in dict_.items():
            if isinstance(attr, Quantity):
                attr.set_name('__'+name, key)


class BusinessEntity(object):
    __metaclass__ = BusinessEntityMeta


class BulkItem(BusinessEntity):
    weight = Quantity()
    price = Quantity()

    def __init__(self, description, weight, price):
        self.description = description
        self.weight = weight
        self.price = price
```

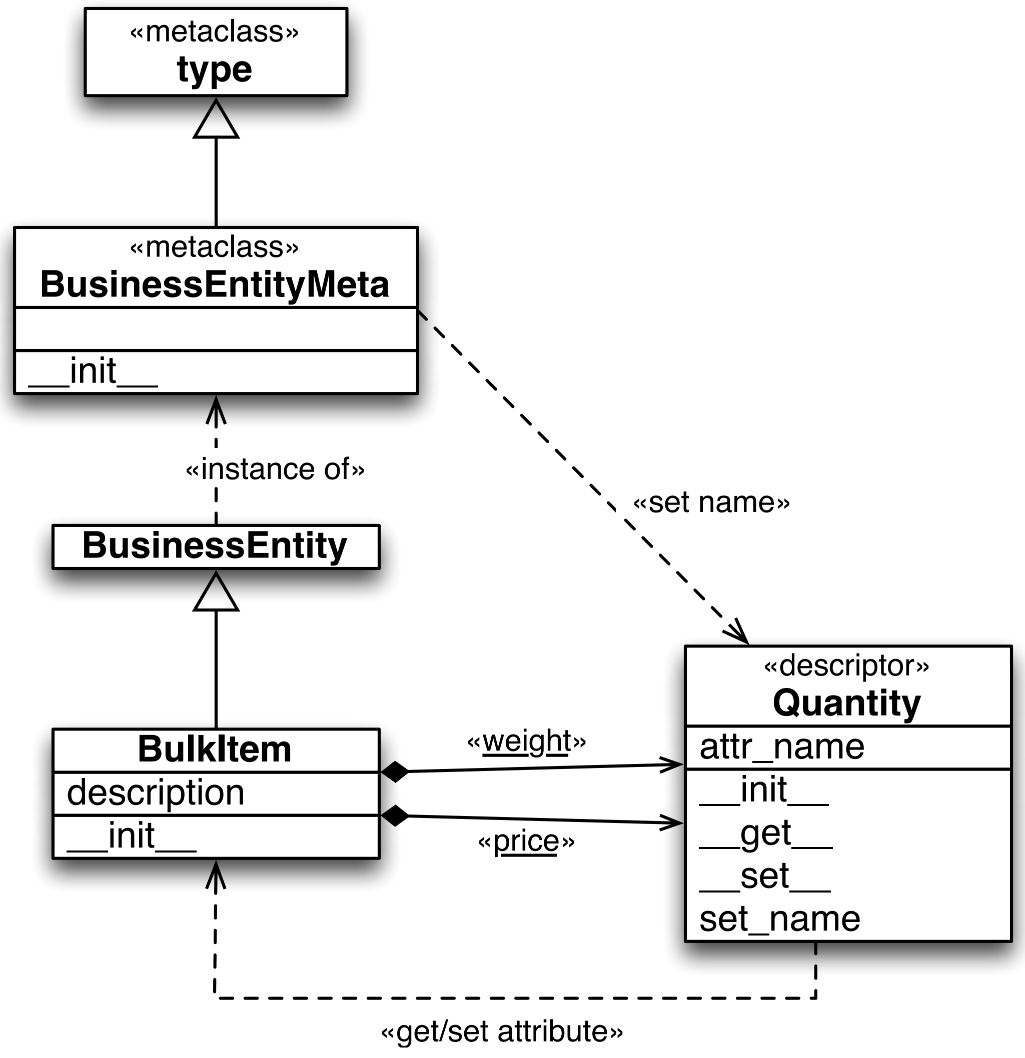
❺ avoiding redundant work



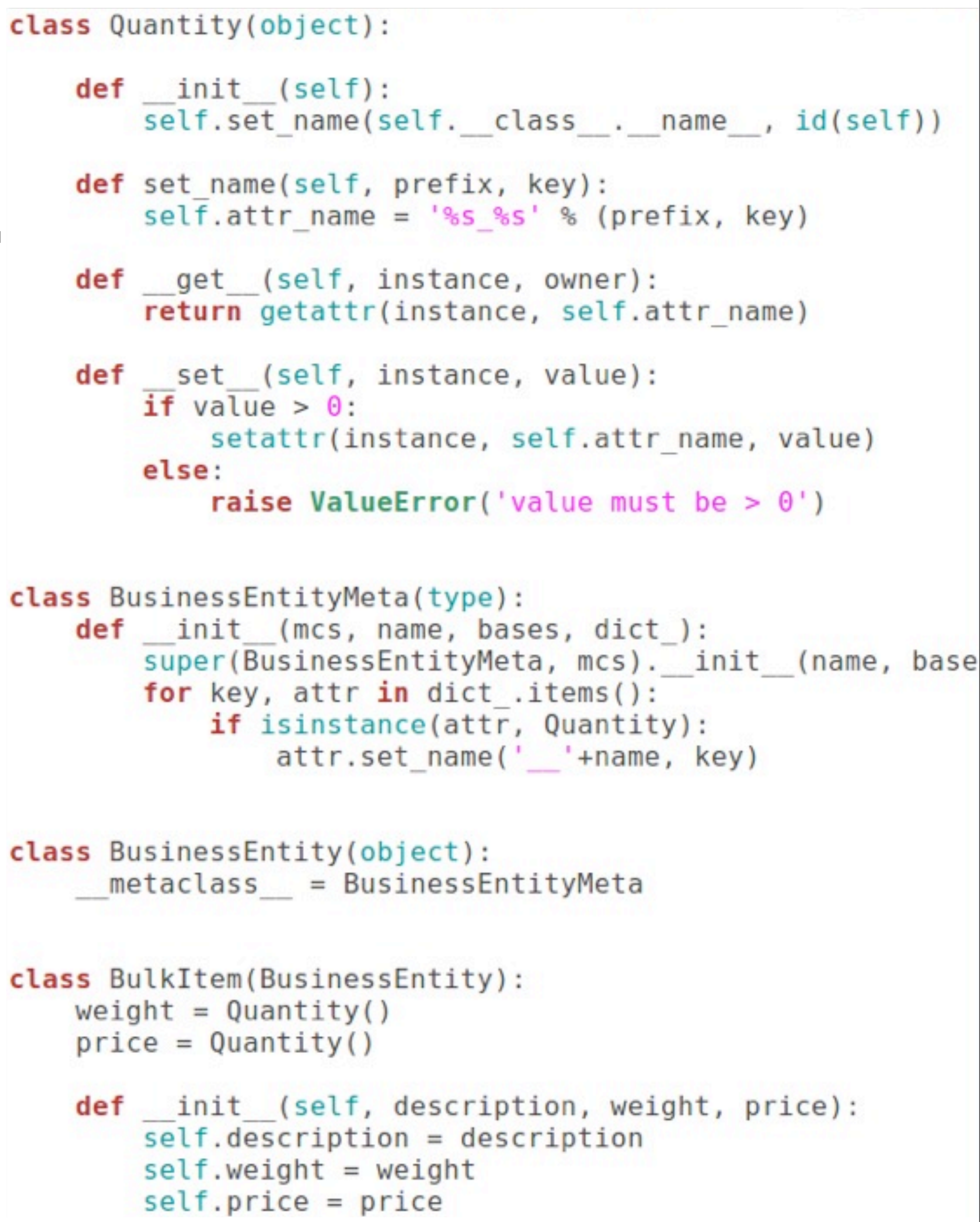
❺ avoiding redundant work



```
class Quantity(object):

    def __init__(self):
        self.set_name(self.__class__.__name__, id(self))

    def set_name(self, prefix, key):
        self.attr_name = '%s_%s' % (prefix, key)

    def __get__(self, instance, owner):
        return getattr(instance, self.attr_name)

    def __set__(self, instance, value):
        if value > 0:
            setattr(instance, self.attr_name, value)
        else:
            raise ValueError('value must be > 0')


class BusinessEntityMeta(type):
    def __init__(mcs, name, bases, dict_):
        super(BusinessEntityMeta, mcs).__init__(name, base
        for key, attr in dict_.items():
            if isinstance(attr, Quantity):
                attr.set_name('__'+name, key)


class BusinessEntity(object):
    __metaclass__ = BusinessEntityMeta


class BulkItem(BusinessEntity):
    weight = Quantity()
    price = Quantity()

    def __init__(self, description, weight, price):
        self.description = description
        self.weight = weight
        self.price = price
```



❺ avoiding redundant work



```
class Quantity(object):

    def __init__(self):
        self.set_name(self.__class__.__name__, id(self))

    def set_name(self, prefix, key):
        self.attr_name = '%s_%s' % (prefix, key)

    def __get__(self, instance, owner):
        return getattr(instance, self.attr_name)

    def __set__(self, instance, value):
        if value > 0:
            setattr(instance, self.attr_name, value)
        else:
            raise ValueError('value must be > 0')


class BusinessEntityMeta(type):
    def __init__(mcs, name, bases, dict_):
        super(BusinessEntityMeta, mcs).__init__(name, base
        for key, attr in dict_.items():
            if isinstance(attr, Quantity):
                attr.set_name('__'+name, key)


class BusinessEntity(object):
    __metaclass__ = BusinessEntityMeta


class BulkItem(BusinessEntity):
    weight = Quantity()
    price = Quantity()

    def __init__(self, description, weight, price):
        self.description = description
        self.weight = weight
        self.price = price
```

The **BusinessEntityMeta** metaclass provides descriptor instances with descriptive attribute names at import time